GENERATION OF REALISTIC TREE BARKS USING DEEP LEARNING

A Dissertation
Presented to
The Academic Faculty

By

Aishwarya Venkataramanan

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer
Engineering

Georgia Institute of Technology

May 2020

Copyright © Aishwarya Venkataramanan 2020

GENERATION OF REALISTIC TREE BARKS USING DEEP LEARNING

Approved by:

Dr. Cédric Pradalier, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. David V Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Paul Voss
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 23rd, 2020

Excellence is a continuous process and not an accident.

Dr. A.P.J. Abdul Kalam

To my parents and to my friends.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Cédric Pradalier for giving me an opportunity to pursue this thesis. I am extremely grateful to him for the continuous guidance and support rendered throughout the thesis. I sincerely thank Antoine Richard for his advice and guidance on the project. I would also like to thank my parents for their continuous support and encouragement. Finally, I appreciate the assistance of all the people who have made this thesis possible.

TABLE OF CONTENTS

Acknowledgments v

List of Figures ix

Summary xii

Chapter 1: Introduction and Background **1**

1.1 Introduction 1

1.2 Introduction to 3D modeling 1

1.3 Related works on 3D modeling 3

1.4 Multi-View 3D Reconstruction. 3

1.4.1 Image acquisition. 4

1.4.2 Camera calibration. 5

1.4.3 Feature extraction and matching 5

1.4.4 Tracking and Mapping. 6

1.4.5 Related work 6

1.5 Generative Adversarial Networks(GANs) 8

1.5.1 Introduction to GANs 8

1.5.2 Conditional GANs 10

1.5.3 Related Works 11
1.6 Conclusion 12

Chapter 2: Technical Approach 13

2.1 Introduction 13
2.2 Data Acquisition 14
2.3 Background Suppression 15
2.4 3D Reconstruction 18
2.5 Surface Geometry Extraction 19
2.6 Generation of fake radius and color image tiles 22
2.6.1 Generating smooth approximate radius maps 23
2.6.2 Generating the bark radius and color using GANs 25
2.6.3 Generating moss and defects at defined places 27
2.6.4 Generating the maps concurrently 32
2.7 Tiling the images 35
2.8 Creating 3D meshes 38

Chapter 3: Experiments 39

3.1 Introduction 39
3.2 Dataset Acquisition for 3D Reconstruction 39
3.3 3D Reconstruction of real trees 40

3.4 Numerical comparison of point clouds 40
3.4.1 Scaling the point cloud 40
3.4.2 Registering the point clouds 41
3.4.3Measuring the distance between the point clouds 41
3.5 Experiment on GANs 42
3.5.1 Dataset 42
3.5.2 Semantic maps for SPADE and Multimodal SPADE 43
3.5.3 Implementation details of pix2pix 44
3.5.4 Implementation details of SPADE and Multimodal SPADE 45
Chapter 4: Results 46
4.1 Introduction 46
4.2 3D Reconstruction and Surface Geometry Extraction 46
4.3 Generative Adversarial Networks 51
4.3.1 Architecture 51
4.3.2 Tiling the images 54
4.3.3 Construction of the 3D meshes 55
Chapter 5: Conclusion **59**

LIST OF FIGURES

1.1 Determining an object point in 3D using triangulation 4

1.2 Estimating the 3D structure of an object from a sequence of images . . . 6

1.3 Architecture of GAN 9

1.4 Architecture of Conditional GAN 11

2.1 Pipeline for generating 3D models of real and fake tree barks 13

2.2 Samples of oak images used for reconstruction 15

2.3 Samples of beech images used for reconstruction 15

2.4 Magnitude of optical flow calculated on video sequences: (a) when the camera translates, the magnitude of the flow is greater in the foreground (b) when the camera rotates, the magnitude of the flow is more in the background 17

2.5 Pipeline for background suppression 18

2.6 3D reconstruction of a tree (a)without background suppression of images (b) with background suppression of images 19

2.7 Radius map with the surface geometry information and the corresponding bark color of an oak tree 21

2.8 2D map of bark color (a) before interpolation (b) after interpolation . . . 22

2.9 Smooth approximate radius maps generated using our method 24

2.10 3D model obtained from a smooth approximate radius map 24

2.11 (a) Input given to pix2pixHD and pix2pix (b) Radius map generated by pix2pixHD (c) Radius map generated by pix2pix. There is a scale variation in the radius values generated by pix2pixHD and the input . . . 25

2.12 Pipeline for generating the radius and bark color 26

2.13 Examples of bark colors with excessive amount of moss and lichens . . 27

2.14 (a) Oak bark generated py pix2pix (b) Beech bark generated by pix2pix. The generated images have no control over the location of moss, bark, and lichens, which looks very unrealistic 27

2.15 SPADE normalization 29

2.16 Network architecture used in [40] 30

2.17 When the input semantic consists of only a single class, pix2pixHD does not produce any meaningful result due to the loss of semantic information after the normalization, while SPADE generates the bark. . 31

2.18 Improved multimodality by using radius maps to generate different oak barks for the same input semantic 32

2.19 The encoder network of Multimodal SPADE to convert input smoothened radius to a latent space 33

2.20 The decoder network of Multimodal SPADE which generates both the radius and color 33

2.21 GAN architecture for tiling images 36

2.22 Tiled image with homogenized color 38

2.23 3D meshes of an oak bark without and with color 38

3.1 Comparison of point cloud from laser scanner with point cloud from (a) Pix4dMapper (b) Colmap. From the histogram in (a) and (b), it can be seen that majority of points have a zero mean in the case of Pix4dMapper, showing that Pix4dMapper yields a better quality reconstruction 42

3.2 Oak bark with the corresponding semantic map 44

3.3 Beech bark with the corresponding semantic map 44

4.1 An oak tree before and after background suppression 46

4.2 3D point clouds of oak trees 47

4.3 3D point clouds of beech trees 47

4.4 3D mesh of an oak tree 48

4.5 Radius maps of oak trees 49

4.6 Oak barks 49

4.7 Radius maps of beech trees 50

4.8 Beech barks 50

4.9 Different methods to generate radius maps for oak trees 52

4.10 Different methods to generate oak bark colors 52

4.11 Different methods to generate radius maps for beech trees 53

4.12 Different methods to generate beech bark colors 53

4.13 (a) The tiled image with discontinuities (b) Image after discontinuities masked (c) Continuous image obtained from the tiling network 54

4.14 Full continuous maps of oak and beech tree barks obtained after tiling . 55

4.15 3D models of oak trees without and with color 56

4.16 3D models of beech trees without and with color 57

4.17 3D models of fake oak and beech bark generated from a smooth approximated radius map 58

SUMMARY

With the increase in demand for high-quality visual content in video games, movies, and simulators, it is of paramount importance to create realistic 3D models of trees, which are ubiquitous and find application in many areas such as urban modeling, movies, and gaming. In this work, we propose a methodology to create realistic 3D models of tree barks using a hand-held camera. There exist many computer graphics techniques which can achieve high quality tree generation; however, only a few works focus on realistic modeling of tree bark. The difficulties in generating realistic tree barks is mainly because of the complex appearance of the bark surfaces. The complexity arises because of the wide variety of barks and their intricate details. Consequently, many methods for realistic tree modeling are being researched. A majority of the work focuses on using the traditional methods in 3D modeling to generate the tree models. In this work, we explore a deep learning based methodology to generate high quality 3D models of tree barks. To the best of our knowledge, this is the first attempt at generating realistic tree barks using deep learning.

We designed a pipeline to generate realistic-looking tree barks using multi-view 3D Reconstruction and Generative Adversarial Networks (GANs). 3D Reconstruction was used to generate tree barks of real trees, while GANs were used to generate tree barks of fake trees.

As part of the pipeline, we developed an efficient method to perform the 3D reconstruction faster and at the same time, generate better quality 3D point clouds. We also analyzed different GAN architectures and loss functions to generate high-quality surface geometry and color of the tree barks. We designed a GAN architecture to generate the surface of the tree barks along with the bark color concurrently. Finally, we developed a GAN architecture to tile small images into a larger and a continuous image.

To test the scope of application of our method, we generated oak and beech tree barks, which have contrasting tree bark structures, using our proposed pipeline. Our experimental results show that our approach generates realistic looking tree barks for both smooth and deeply ridged surfaces. The generated tree barks look realistic for both the tested bark types and our method was able to capture the fine details in the tree bark surface. Ultimately, this method could be used to generate thousands of 3D tree bark models for several types of trees.

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Introduction

The objective of the project is to create 3D models of real trees and use those models to generate fake trees, with tree defects and moss at defined locations. The problem can be divided into two major steps:

1. 3D reconstruction of real trees to obtain their geometry and color.

2. Creating fake trees, to generate more examples of geometry of trees using GANs.

In the coming sections, we will review the basics of 3D modeling and the existing methods proposed by the research community to generate 3D models of trees. We will also present a brief overview of multi-view 3D reconstruction. Since we use GANs to generate fake 3D models using the real 3D models for training, the basics of GAN will be explained and the existing state-of-the-art GAN architectures and their applications will be introduced.

1.2 Introduction to 3D modeling

3D modeling dates back to the 1960s, when Sketchpad, the first 3D modeling software was invented by Ivan Sutherland, while at MIT. Since then, 3D models have been widely used in computer graphics and CAD. With the advances in the modeling techniques and the development of computer hardware, the quality of the models improved over time. Eventually, 3D modeling found its applications in various industries like animation, gaming, architecture, and interior designing. Over the years, several techniques have evolved and the 3D modeling softwares have undergone a drastic improvement, with the addition of many features for users to easily create 3D models.

3D modeling can be broadly categorized into three major approaches:

1. Procedural modeling,

2. Sketch-based modeling,

3. Data-driven modeling

Procedural modeling uses a set of rules and algorithms to generate 3D models. Generally, procedural modeling requires the user to fine-tune a set of parameters to create a model. However, defining the model and their parameters is not an easy task and is time-consuming. L-Systems, and fractals are examples of procedural modeling techniques.

In **sketch-based modeling**, a 2D sketch of the model is created which is then converted into 3D using an application. Sketch-based modeling is primarily designed for use by persons with artistic ability, but no experience with 3D modeling programs. However, this method is a tedious task when the object to be modelled contains a lot of textures and intricate details. Hence, this method is mostly restricted to rapid modeling of low-detail objects for use in prototyping and design work.

In a **data-driven** approach, an object is scanned and a 3D model is reconstructed from the scan. With high-quality reconstruction, the 3D models capture the nuances in the geometry which are required for the model to be realistic. Laser scanners, structured light scanners, tomography, photogrammetry are some of the commonly used methods to obtain a 3D model of an object from reconstruction. 3D Laser scanners provide accuracy in the range of few millimetres. Given the high cost of laser scanners and tomography machines, a cost effective method would be to capture images of the tree from different positions and 3D reconstructing it, also known as multi-view 3D reconstruction or photogrammetry. Unlike a laser scanner, this method can also capture the color information of the object being reconstructed, along with the geometry.

For our study, we use photogrammetry to obtain the 3D reconstruction of the trees. The tree is modelled by capturing images of the tree from different camera poses and using the captured images for the 3D reconstruction of the trees.

1.3 Related works on 3D Modeling of Trees

Tree modelling is an important aspect of computer graphics because of its wide range of applications in computer generated scenes. However, given the numerous varieties of trees and complex bark structures, it is challenging to achieve a high level of realism. Furthermore, having a wide diversity of texture for the same tree species is equally challenging. The existing methods fall into one of the three major 3D modelling approaches: procedural, sketch-based and data-driven.

There are several previous works on generating tree models along with branches and leaves[1-6]. [1] uses tree-cuts from real tree models to generate fake trees and [2,3] use sketch-based modeling. However, very few works focus on generating realistic tree barks. [4] uses X-ray images of a real bark to generate bark structure, while [5] uses texton analysis to generate barks from a single photograph. [6] uses procedural modeling to generate barks.

We use the data-driven method for modeling the tree, by capturing image sequences of the trees from different poses and reconstructing the tree from the images. The principle behind that is multi-view 3D reconstruction. In the next section, a brief overview of the 3D reconstruction using multiple images is given.

1.4 Multi-View 3D Reconstruction

In multi-view 3D reconstruction, a set of images of an object, taken from multiple camera poses is used to create a 3D model of the object. When an image of an object is captured, the 3D scene is projected onto a 2D plane and the depth information is lost in the process. A single point on the 2D plane corresponds to all the points on a line of sight in 3D. Hence, the exact point in 3D cannot be determined from a single image. If two images are present, then

the point in 3D can be determined by a process known as triangulation. In triangulation, the position of a point in 3D can be determined by the intersection of two projection rays from the images. Figure 1.1 illustrates the triangulation principle based on stereo (from two images). Using this principle, the relative scale and depth information can be obtained from the set of images and used to construct a 3D point cloud of the scene.

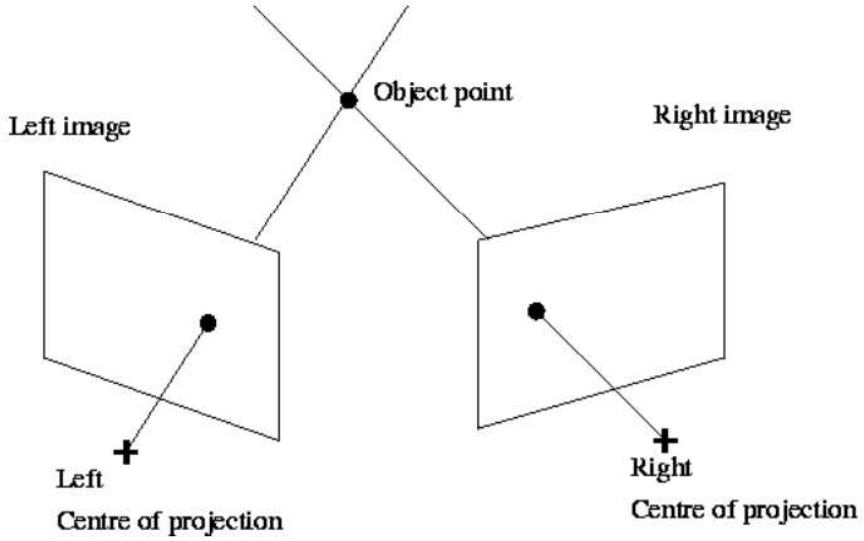


Figure 1.1: Determining an object point in 3D using triangulation. Source:

[http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT10/node3.html]

The 3D Reconstruction process can be broken down into the following steps:

1.4.1. Image acquisition

A minimum of two images, taken from different positions are required to capture the depth information of the object. To obtain the complete 3D model, images of the object are taken from multiple positions, which includes the region to be reconstructed. To yield a high quality 3D reconstruction, the images must be captured using a high resolution camera and the images must be free from noise and motion blur.

1.4.2. Camera calibration

Camera calibration is done to obtain the parameters (camera matrix) which can be used to map the relationship between the 2D image co-ordinates and the corresponding points in 3D world.

The relation between the image co-ordinate and the 3D co-ordinate is given by

$$W[x\ y\ 1] = [X\ \ Y\ \ Z\ \ 1]P$$

where, W is the Scale factor, $[x\ y\ 1]$ are the image points, $[X\ \ Y\ \ Z\ \ 1]$ are the world points, P is the camera matrix

The camera matrix is further represented as a product of two matrices,

$$P = \begin{bmatrix} R \\ t \end{bmatrix} K$$

where, $\begin{bmatrix} R \\ t \end{bmatrix}$ is the Extrinsic matrix (rotation and translation of the camera), and K is the Intrinsic matrix

The world points are transformed to camera coordinates using the extrinsics parameters. The camera coordinates are mapped into the image plane using the intrinsics parameters.

1.4.3. Feature extraction and matching

Once the images are obtained, the next step is to extract feature points that are common between two or more images. SURF[55], SIFT[56], FAST[57], ORB[58] are some of the commonly used feature detection methods.

Once the feature points are detected, they must be matched to find the correspondence between the images. Feature matching should be robust against illumination changes, noise, etc.

1.4.4. Tracking and Mapping

To build a complete 3D model in real-time using images, one needs to estimate the camera position and at the same time build the 3D model. If the position of the camera is

known, then the 3D co-ordinates of the points of the object can be estimated. Likewise, if the 3D surface of the object is known, the camera poses can be estimated from it. However, neither the camera pose nor the 3D co-ordinates of the points are known beforehand in real time. The general solution is to simultaneously estimate the position of the camera and map the points in 3D. This is known as visual SLAM. Figure 1.2 illustrates the process of reconstructing the 3D model of an object from a given sequence of images.

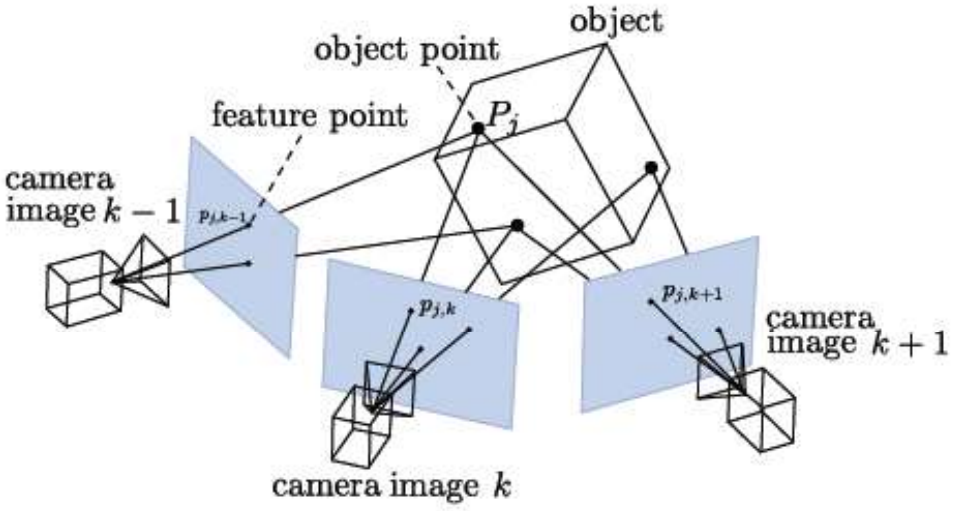


Figure 1.2: Estimating the 3D structure of an object from a sequence of images
Source: [openmvg.readthedocs.org/en/latest/_images/structureFromMotion.png]

1.4.5 Related work

There exist many methods in the literature to perform 3D Reconstruction. Visual SLAM is divided into two approaches: feature based and feature-less. Feature based methods utilize feature detectors and descriptors to perform the tracking and mapping. In contrast, feature-less methods directly use the input images to perform tracking and mapping. In general, photometric consistency is used as an error measurement in direct methods whereas geometric consistency such as positions of feature points in an image is used in feature-based methods.

Feature based methods

The first monocular visual SLAM was developed in [7] and is called monoSLAM. MonoSLAM uses Extended Kalman filter to estimate the camera poses and the 3D structure of the environment. The drawback with monoSLAM was the high computation cost, which was proportional to the size of the environment. To solve the problem of computation cost in monoSLAM, Parallel Tracking and Mapping (PTAM) [8] was introduced. In [8], the reconstruction is done by splitting tracking and mapping into two separate tasks and the tasks were processed in parallel threads. Since the tracking and mapping were split into separate threads, PTAM could handle much more feature points compared to monoSLAM and was also computationally efficient. ORB-SLAM [9-10] is an extension of PTAM, which includes vision based close-loop detection in addition to the parallelized tracking and mapping.

Feature-less methods

DTAM [11] is a fully feature-less method, where the tracking is done by comparing the input image with synthetic view images generated from the reconstructed map. DTAM uses a hand-held RGB camera to create a dense 3D surface model and uses it for dense camera tracking. In LSD-SLAM [12], reconstruction is limited to areas which have an intensity gradient, which means, it ignores texture-less areas since it is difficult to estimate depth accurately from images in those areas. DSO [13] is a fully direct method, which removes error factors as much as possible from geometric and photometric perspectives. [14,15] use deep learning to estimate the camera pose and the depth.

Recently, structured light-based RGB-D cameras such as Microsoft Kinect are becoming cheaper and smaller. These cameras provide both the color images and dense depth maps in real-time. [16] proposes an approach to perform SLAM using a RGB-D camera.

In this study, we use the photogrammetry softwares Colmap [17,18] and Pix4DMapper [19], which are feature based methods, to obtain the 3D reconstruction of the trees using an RGB camera.

Once the 3D point cloud of the trees is obtained from reconstruction, we extract the surface geometry information and the bark color from the point clouds, and store them as 2D maps. Finally, we use GANs [19] to generate fake 2D maps. The fake maps are then used to construct a 3D model of the tree. The next section gives a brief overview of GANs.

1.5 Generative Adversarial Networks (GANs)

Since the advent of AlexNet [20] and its impressive performance on the ImageNet dataset, deep learning techniques have attracted wide-spread attention in both the academic and industrial community. Deep learning has found its application in many of the computer vision tasks and are widely used in image classification, object detection, image segmentation, image synthesis, etc. With gradual improvements in the network architectures and robust loss functions, deep learning methods have surpassed human capabilities in many tasks and are being used in medical imaging, automated driving, natural language processing, etc. The development of powerful GPUs mainly contributed to the improvements in the network architectures.

Inspired by the success of deep learning techniques in many of the computer vision tasks, we wanted to explore its applicability for our objective: modeling tree barks, and assessing the quality of the 3D models obtained using deep learning.

1.5.1 Introduction to GANs

Generative Adversarial Networks (GANs) [19] belong to the class of generative models, where deep learning is used to generate new content. The advent of GANs has brought in a lot of interesting applications and impressive performance in image generation and speech

synthesis tasks. GAN was introduced in [19], where image generation from an input noise image was demonstrated on MNIST, TFD and CIFAR-10 datasets. A GAN consists of two networks: a generator and a discriminator acting as adversaries to each other. A typical GAN architecture is given in Figure. 1.3.

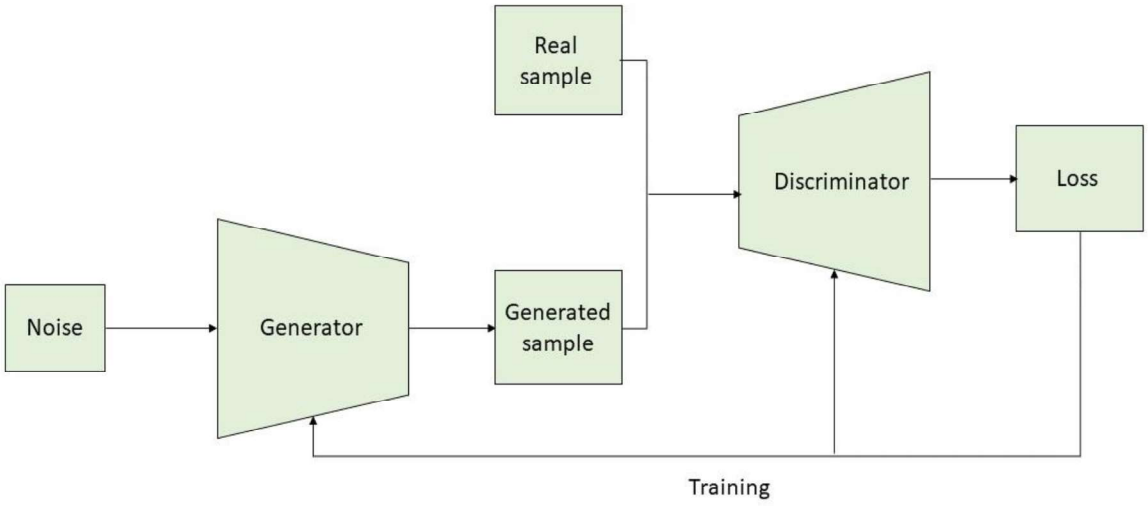


Figure 1.3: Architecture of GAN

The function of the generator network is to produce samples such that the discriminator cannot differentiate between the training data and the generated data. The function of the discriminator is to correctly classify if the sample provided to it is from the generator or the training data. As the training progresses, both the networks become good at their own task. The generator produces samples that are close to the real data and the discriminator correctly identifies the fake image from the real. At one point the generated image looks realistic so that the discriminator cannot correctly identify if the sample came from the generator or the training data. At this point the training is complete.

The loss function for training a GAN network is

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[log(D(x))] + E_{x\sim p_z(z)}[log(1 - D(G(z)))]$$

where, G is the generator and D is the discriminator, $p_{data}(x)$ is the prior on the training data, and $p_z\,(z)$ is a prior on the input noise variables. $D(x)$ represents the probability that x came from the data rather than from the generator. $D(G(z))$ represent the probability that the sample is from the generator.

1.5.2 Conditional GANs

The more popular usage of GANs is in a conditional setting to generate meaningful images, given a conditioned label. The labels could be discrete class labels, semantic maps, edge maps, reference images, etc. This type of network is known as a Conditional GAN (cGAN). In a conditional GAN, a label is given as input to the generator and the discriminator network. The labels contain some underlying information on the kind of image to be generated. The conditioned labels help the GAN train faster and produces a better quality output compared to the normal GAN. The architecture of Conditional GAN is given in Figure 1.4.

Conditional GAN was introduced in [22], where they demonstrated the generation of hand-written numbers with input class labels encoded as one-hot vectors. Since then conditional GANs have been used on discrete labels [22, 23], text [24] and images [25,26] to generate meaningful images.

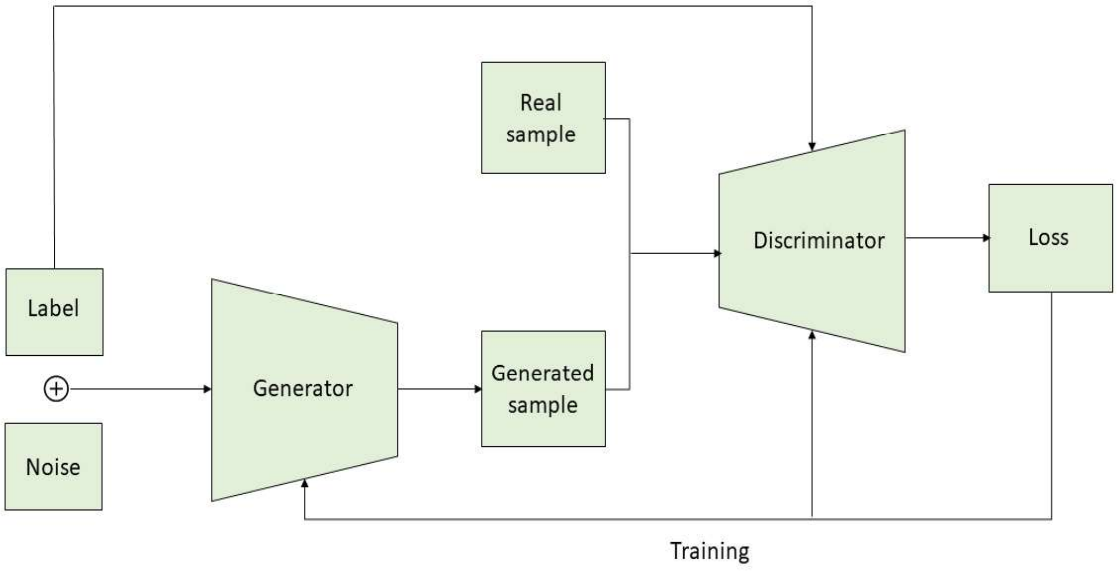


Figure 1.4: Architecture of Conditional GAN

1.5.3 Related Works

The original GAN paper used neural networks to generate the images. Later, DCGAN [26] was introduced, which incorporated Deep Convolutional Neural networks in the generator and discriminator to generate images. Using deep CNNs boosted the quality of the generated images compared to using neural networks. Since then, novel loss functions [28,29] and deep CNN architectures [30-32] have been evolving to stabilize training and generate higher quality images. Recent developments have enabled the generation of high resolution images of human faces [30-32].

With the introduction of pix2pix [33], conditional GANs attracted wide-spread attention because of its ability to generate high quality images. Pix2pix is an image translation network, where the input label is an image and the network generates an image corresponding to the input. Various image-to-image translation networks have been proposed in [34-36, 39,40] since the success of pix2pix. CycleGAN [34] is used when an unpaired dataset of input

and target images are available for training. StarGAN [35] is used for multi-domain image-to-image-translation. BigGAN [36] is used to generate high quality natural images. Pix2PixHD [39] uses multi-scale generator and discriminator network to generate high resolution images from input semantic maps, while SPADE [40] uses a spatially adaptive normalization to retain the semantic information through the network.

GANs have found application in data augmentation [37], image-to-image translation, text-to-image translation [38], converting semantic label map to photo-realistic images [39, 40], image inpainting [41, 42], image blending [43] to name a few.

1.6 Conclusion

Data driven modeling is being widely used nowadays because of its simplicity compared to other modeling types and its ability to capture fine details. Variational auto-encoders and GANs are the two commonly used deep learning architectures to generate images. When trained properly, GANs can achieve high level of realism in the generated images. Thus, we use GANs to generate fake samples of tree barks. For our objective, we combine the easiness of data-driven modeling and the ability of GANs to generate high quality images to our advantage to generate 3D models of realistic looking barks.

CHAPTER 2

TECHNICAL APPROACH

2.1 Introduction

In this chapter, we explain the pipeline to generate realistic tree barks from a sequence of images. The block diagram of the pipeline is shown in Figure 2.1. Each of the blocks will be explained in more details in the coming sections. We developed the pipeline to generate fake oak tree barks and later on extended it to generate beech barks. Thus, the proposed pipeline can be used for a large variety of tree barks.

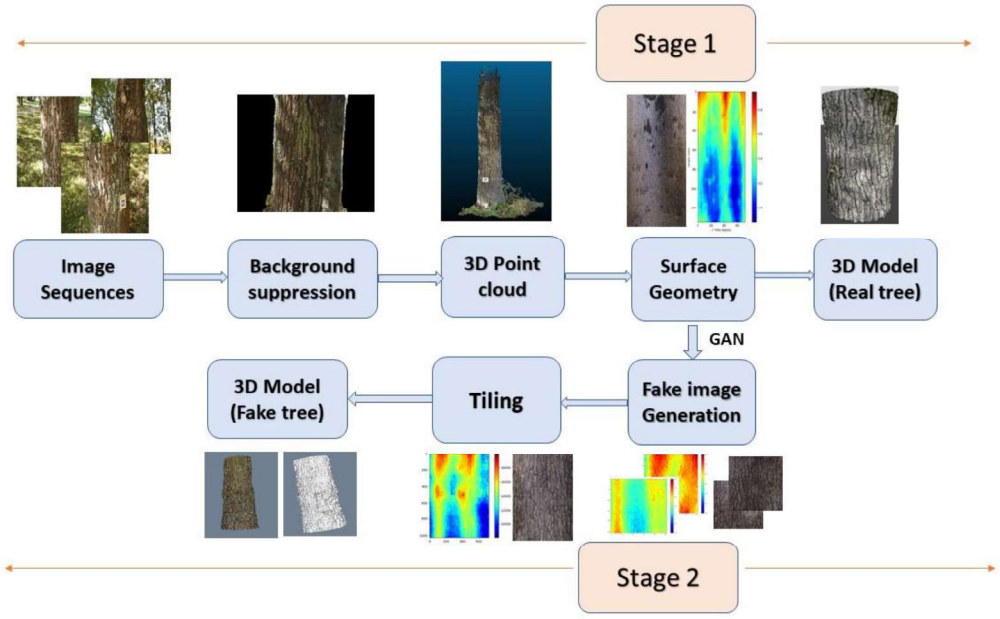


Figure 2.1: Pipeline for generating 3D models of real and fake tree barks

2.2 Data Acquisition

As shown in Figure 2.1, the first step in the pipeline is to capture image sequences of trees. To perform 3D reconstruction and triangulation, the image sequences of trees must be captured from multiple camera positions. Initially, while testing with normal hand held cameras, motion blurs were introduced due to jerking of the hand. This further deteriorated the quality of the 3D reconstruction. To avoid this, DJI Osmo pocket, a gyro-stabilized camera was used which could also capture images in Full HD. Gyro-stabilization gives sharper images by counteracting the camera shakes.

Since we capture the images using a monocular camera i.e a single camera is used to capture all the images, the global scale information of the tree is not preserved in the final 3D point cloud. To get the scale information, we attach markers with known dimensions onto parts of the trees while capturing the video/images. We placed the markers onto the part of the tree near the ground so that it does not obstruct the mid portion of the trunk, which is required for extracting the surface geometry and color information. Once the 3D point cloud of the tree is obtained from reconstruction, the known marker dimension is used to scale the tree accordingly.

Videos of 20 oak and 15 beech trees were captured from a local forest area, under natural lighting. Since natural lighting was used, our dataset consists of parts of tree barks illuminated by sunlight as well as dark regions because of shadow. Some of the images of the oak tree with uneven illumination are shown in Figure 2.2. Figure 2.3 shows the images of the beech trees.

Once the videos of the trees were acquired, the image sequences were obtained from all the video frames. Each video consists of thousands of image frames. The running time of the photogrammetry software depends on the number of input images given to it. Using all the

images from a video sequence slows down the software drastically and requires a lot of computing power. Since we used a gyro-stabilized camera, the transition between the consecutive frames were smooth and hence, there were no major changes between consecutive frames. Most of the consecutive frames contain redundant information and so, the images were extracted from every tenth frame of the whole video sequence. This way, the number of images were in the range of few hundreds, which could be handled efficiently by the photogrammetry softwares.

Figure 2.2: Samples of oak tree images used for reconstruction

Figure 2.3: Samples of beech tree images used for reconstruction

2.3 Background Suppression

Once the image sequences of the trees were obtained, they were used to reconstruct the trees. The photogrammetry softwares Colmap[17, 18] and Pix4DMapper[19] were used, which uses the image sequences to produce 3D models of trees. These are feature based methods and as explained in Section 1.3. 3D reconstruction in feature based methods require feature points

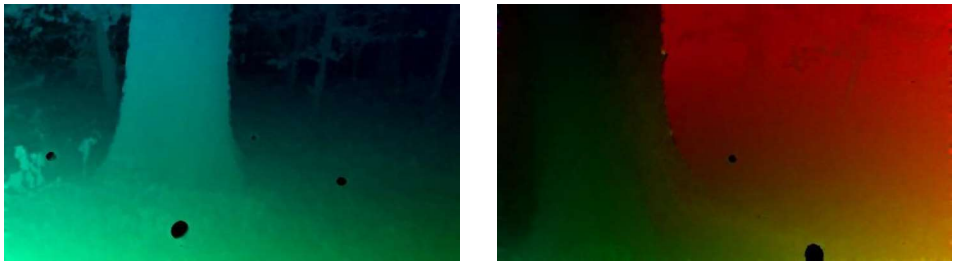
that are common between two or more images to be extracted and matched. So, if a same feature point occurs in any two images, it will be reconstructed. This means, apart from the tree that needs to be reconstructed, several other points in the background will also be reconstructed. This yields a sparser reconstruction of the tree and necessitates cleaning up the unwanted points later. For the study, we are only interested in the 3D point cloud of the tree, hence, we preferred the 3D reconstruction to focus more on the tree rather than the surrounding. Removing the background alleviates this and also makes 3D reconstruction run faster.

To remove the background portions from the images, the deep learning segmentation network, proposed in [44, 45] was finetuned from Cityscapes [46] to label the bark and the non-bark regions. We chose PSPNet [44] as it utilizes global scene clues to segment the image. This translates to less noisy and "hole free" segmentation maps when compared to more traditional segmentation networks. To train the PSP network, the images and the corresponding labels indicating the tree bark and the surrounding regions are required. Acquiring a labeled dataset to train a segmentation network is a challenging task, since most of the time, labeling has to be done manually. As our images were acquired with a gyro-stabilized camera, we could assume that the transition between the consecutive frames is smooth. Using this assumption, we developed an automated labeling mechanism based on optical flow to acquire a labelled dataset to train the deep learning segmentation network.

Optical flow was considered to differentiate the foreground (tree bark) from the background of the images. First, we obtained the horizontal and vertical flow vectors for all the consecutive images in the video sequence. From the flow vectors, we calculated the magnitude and direction of the flow. As our images are acquired with a gyro-stabilized camera, we can make the two following hypotheses. When the camera translates along the tree, the flow direction is vertical and the magnitude of the flow vectors are more in the foreground than in the background since the objects close to the camera move faster than the far-away objects.

When the camera rotates, the flow direction is horizontal and the magnitude of the flow vectors is more in the background since far-away objects move faster. A sample of the flow magnitude when the camera translates and rotates is shown in Figure 2.4.

An adaptive threshold on the magnitude of the flow vectors was used to mask the background. The magnitude of the maximum flow vector for each frame was taken and compared with the other flow vectors for the same frame. As the camera translates, and the magnitude of the flow vector is greater than 50% of the maximum magnitude, the region was labelled to be the foreground. The other regions were labeled as background.



(a) (b)

Figure 2.4: Magnitude of optical flow calculated on video sequences: (a) when the camera translates, the magnitude of the flow is greater in the foreground (b) when the camera rotates, the magnitude of the flow is more in the background

The labelled images were used to train the segmentation network. Once trained, the original images acquired from the videos were provided as input to the network. The output from the segmentation network is a label indicating the bark and the non-bark regions for each of the input images. The output label images were used to create a mask, where the bark portions were represented as 1s and the other regions as 0s. The segmentation network sometimes detects some of the trees in the background and marks them as 1. To remove those trees, the largest connected region, which is the tree bark to be reconstructed, was detected in

the mask and retained, while the other regions were erased. The final mask obtained was dilated to include the edges of the tree bark. The mask obtained was applied to the original image to get the tree bark image with the background portions suppressed. The pipeline for background suppression, showing the intermediate steps is given in Figure 2.5.

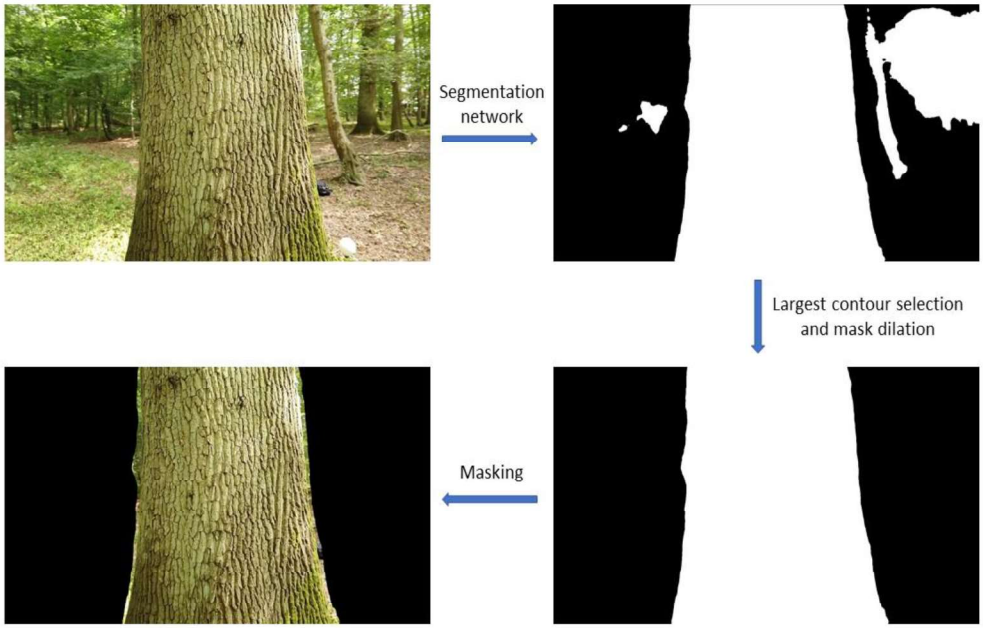


Figure 2.5: Pipeline for background suppression

2.4 3D Reconstruction

The images, with the background portions suppressed, were used by Colmap and Pix4d to generate a dense 3D point cloud of the tree. Figure 2.6(a) is a 3D reconstruction of a tree without performing background suppression. It can be seen that the surrounding is also reconstructed, which results in a sparser reconstruction of the tree. Figure 2.6(b) is a 3D point cloud of a tree obtained after background suppression. The point cloud obtained is denser and cleaner. Since after removing the background portions from the images helped the

photogrammetry software focus only on reconstructing the tree, it was also able to capture the structural details like the ridges in oak trees. Hence, background suppression of the images proved to be beneficial by yielding a better quality point cloud and reducing the work needed to clean up the surroundings in the point cloud.

Figure 2.6 3D reconstruction of a tree (a)without background suppression of images (b) with background suppression of images

2.5 Surface Geometry Extraction

Once the 3D dense point cloud of the tree is obtained, we extracted the surface geometry information from the point cloud, along with the bark color. To do this, we approximated the 3D point cloud of the tree to a series of circles stacked vertically. In the cylindrical co-ordinates, the point cloud is discretized as height and angle. The height was discretized in steps of 1 mm and angle in steps of 0.5°. For each of the circles obtained from the discretized height, the centre point was calculated using a circular least-square fit. Initially, RANSAC was used to find the centre points. However, the centre calculation using RANSAC was inaccurate. Since the shape of the tree bark is not exactly a circle, RANSAC filters most of the points as outliers and chooses only a small section as inliers and calculates the centre point based on the chosen

"inliers". This resulted in incorrect radius calculation. Thus, we used circular least-square fit for our centre-points calculation.

A 2D map was constructed with radius, measured from the center point as a function of angle and height. Since the height and angle are discretized, there may be many points corresponding to a particular value of height and angle. So we take the mean values of the radius of the points to construct the radius map. The mean radius is calculated as a gaussian weighted sum of the distance between the point and the center of the cell in the map divided by the sum of weights. The mean radius calculation also includes the points from the neighbouring cells. The formula used to calculate the mean radius is given below:

$$\text{Mean radius} = \frac{\sum w_i r_i}{\sum w_i}$$

where, w_i is the gaussian weight, given by

$$w_i = e^{\frac{-|p-c_k|}{2\,\sigma}}$$

p is the point from the point cloud, c_k is the centre of the cell for which the mean radius is being calculated.

Similarly, the mean values of R, G and B are calculated as a function of height and angle, where instead of the radius the R, G, B values of the point is used. A sample radius and the color map is shown in Figure 2.7. The radius map is shown in jet color for visualisation.

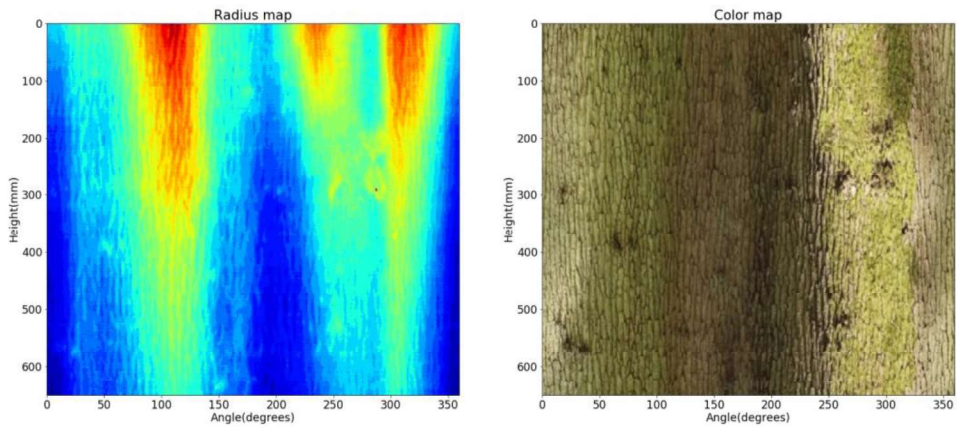


Figure 2.7: Radius map with the surface geometry information and the corresponding bark color of an oak tree

Sometimes, there may be cells in the map where there are no points. This is much more prevalent in the case of oak trees since the ridges on the bark surface creates occlusions, which are not reconstructed in the point cloud. Since the bark of the beech tree is much smoother, the occlusions are minimal. The presence of the cells with no points create empty regions in the final map. To avoid this, a seven-level image pyramid is constructed. The lowest level in the pyramid consists of the original resolution map and the highest level consists of map with resolution $\frac{1}{2^7}$ of the original map. For each level in the pyramid, if there are cells with no point, the radius and RGB values are interpolated from the map of the next level up. Figure. 2.8 demonstrates the need for interpolating. Figure 2.8(a) is a 2D map of an oak bark before interpolation. The black regions in the image are due to the empty cells. Figure 2.8(b) is after interpolation. The empty cells are filled with the values from the higher level maps in the pyramid.

The radius values are typically in the range 0-1, so they were multiplied by 255 to store them as uint8 image. However, when stored as uint8 image, the radius values were compressed and the precision was lost. So the radius values were multiplied by 65535 and stored as uint16 images to retain the precision.

(a) (b)

Figure 2.8: 2D map of bark color (a) before interpolation (b) after interpolation

2.6 Generation of fake radius and color image tiles

One of the objective of our work is to create 3D models of fake tree barks, for which we use deep learning (GANs). As introduced in Section 1.5, GANs are a class of generative model that uses deep learning to generate new data based on the distribution of the training data. To generate new tree bark models, we train a conditional GAN with the radius and color maps obtained from real trees. Since GANs could produce good quality results only at a small resolution, we divided the originally obtained maps into small tiles and used them for training. Our aim is to generate the fine surface details and the color of the bark from a smooth surface of the bark. To achieve this, we divide the process in two steps. In the first step, a GAN is trained to generate the radius map from a smooth surface map. In the second step, another GAN

was used to generate the bark colors from the radius map. To generate fake trees, we need a method to generate the smooth bark surface, on top of which the GAN generates the finer bark details. We now present the mathematical model to generate smooth approximate maps, which can be approximated to the smooth bark structure of real tree trunks. Using this method, thousands of unique smooth approximate tree trunks can be generated.

2.6.1 Generating smooth approximate radius maps

The smooth approximate surface of the bark represents the overall structure of the tree without any of the finer structure details. The map consists of radius values r of the tree trunk as a function of height z and angle, similar to the radius map shown in Figure 2.7, but without the finer bark structures. For training the GAN, we smoothen the radius maps of the real trees using a Gaussian filter. However, while testing, we generate smooth approximate radius maps. To generate these maps, initially, we create a 2D map with an exponentially decaying radius r_0.

$$r_0 = ax^z$$

where,

x is a floating point number between the range 0.8 and 0.9, and a is a random value chosen between 0.1 and 0.9.

To approximate the wavy structure of an actual bark, add a series of sinusoids of varying amplitude and frequencies to r_0.

$$r = r_0 + \sum_{i=1}^{n} b_i \sin(\omega_i \theta_i)$$

where,

n is a random integer chosen between 1 and 3

b_i is the amplitude of the i^{th} sinusoid

ω and θ are chosen to ensure that r is continuous between 0 and 2π.

The amplitudes and frequencies of the sinusoids, and the value of a are randomly chosen so as generate unique smooth tree barks every time. Using this method, any number of smooth tree barks can be generated. The obtained smooth approximated radius maps are provided to the GAN, which generates the finer tree bark structures on top of the smooth maps. Some of the smooth approximated radius maps obtained using this method are given in Figure.2.9, and a 3D model of one of the smooth approximate bark is given in Figure. 2.10.

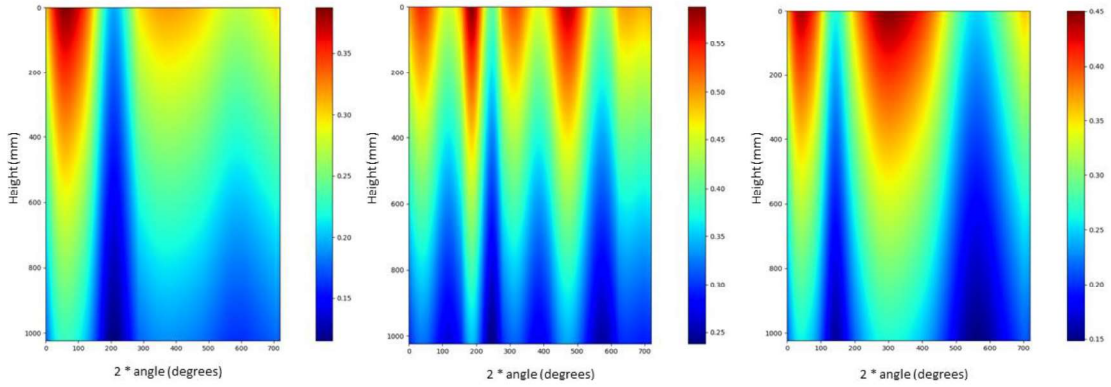


Figure 2.9: Smooth approximate radius maps generated using our method

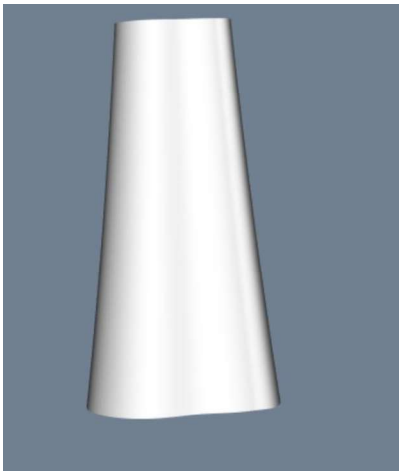
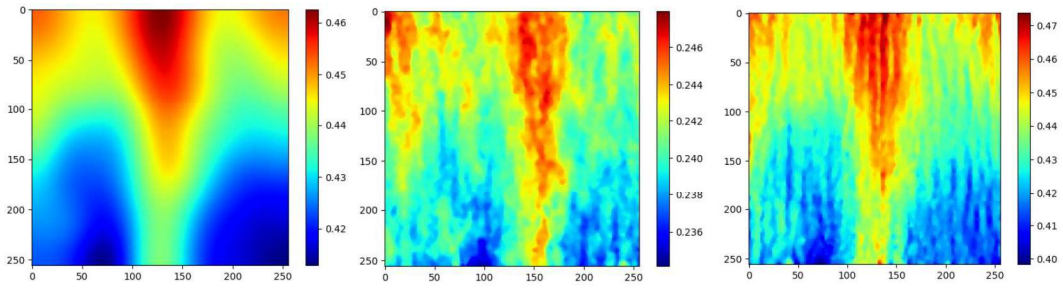


Figure 2.10: 3D model obtained from a smooth approximate radius map

2.6.2 Generating the bark radius and color using GANs

Once the smooth surface of the barks are obtained, the next step is to generate the fine bark structure and the bark color, on the smooth bark surface. We initially considered two of the state-of-the-art conditional GAN architectures: pix2pix[15] and pix2pixHD[39] for generating the images and later on moved on to more sophisticated networks to overcome the shortcomings of these architectures. Pix2pix could produce good results only up to a maximum image resolution of 256×256. Next pix2pixHD was considered, which is capable of generating high resolution images. However, when tested on our dataset, there were color scale variations observed between the input and the generated images. This was not desired for the radius map generation, since the scale variation destroys the relation between the input and the generated maps. An example is shown in Figure 2.11. When the input radius values are in the range 0.40-0.47, pix2pixHD generates radius values in the range 0.23-0.25, whereas pix2pix maintains the input radius range while generating. The scale variations observed in pix2pixHD was also found to be random. The pix2pixHD network also generated a lot of artifacts. Considering the above mentioned factors, pix2pix[15] was chosen to generate the fake radius and color maps.



(a) (b) (c)

Figure 2.11: (a) Input given to pix2pixHD and pix2pix (b) Radius map generated by pix2pixHD (c) Radius map generated by pix2pix. There is a scale variation in the radius values generated by pix2pixHD and the input.

Since pix2pix is capable of generating high quality results up to a resolution of 256×256, the input images to the network are divided into tiles of resolution 256×256. The radius maps, unlike color maps contain pixel values only within a limited range. So the entire set of radius maps were normalised to the range 0 and 65535. The images were further instance normalized i.e the pixel values were converted to the range [-1,1] before feeding them to the network.

Once the normalized smooth approximate maps and the radius maps are obtained, we use two pix2pix networks, the radius map generator and the color map generator to generate the bark radius and color. The radius map generator generates radius maps with the finer bark details from the input smooth approximated radius map. The generated radius maps are then used by the color map generator to generate the corresponding color map. Our pipeline for generating the surface geometry and color is shown in Figure 2.12.

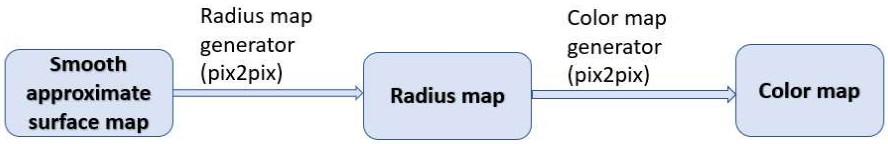


Figure 2.12: Pipeline for generating the radius and bark color

The method works well for generating the surface map, however the generated color maps have moss and bark regions mixed up, which makes the bark images appear unrealistic. Our dataset contains trees with moss, lichens and defects over the bark surface, given in Figure 2.13. The GAN learns to generate them on the fake trees. But there is no control over their location and the regions where they occur. Example of an oak and beech bark generated by pix2pix is given in Figure 2.14. This instigated us to explore other GAN architectures which would allow us to constrain the color map generated.

Figure 2.13: Examples of bark colors with excessive amount of moss and lichens.

(a) (b)

Figure 2.14: (a) Oak bark generated by pix2pix. (b) Beech bark generated by pix2pix. The generated images have no control over the location of moss, bark, and lichens, which looks very unrealistic.

2.6.3 Generating moss and defects at defined places

The moss generated by the GAN network at random places creates an unrealistic effect, since moss grows predominantly on the north side of the tree in the Northern Hemisphere and on the south side in the Southern Hemisphere. Hence, to generate realistic looking bark color,

we constrain the GAN network further, where we also have an additional input to the bark color generator network. We provide a semantic label map indicating the location of the moss, defect, bark and lichens to the network.

The label map is a single channel image with class numbers assigned for each class of object in the image. The label maps were obtained by hand labeling the color map. The color maps were classified into the following four regions: bark, moss, defect, and lichens, with label values of 0 to 3 for the four classes. Once the label maps were prepared, they were used by GAN to generate the color maps. The state-of-the-art GAN network to translate label maps to photorealistic images is SPADE[40]. SPADE is an extension of pix2pixHD[39], with a modified generator architecture.

Pix2pixHD uses a coarse-to-fine generator and a multi-scale discriminator combined with robust objective functions to generate high-resolution images. The pix2pixHD network generates impressive results when provided with label maps with multiple classes, even for high resolution images. However, it fails to generate a meaningful image when provided with a label map consisting of a single class. The reason being, when a semantic map with a single value is passed through a convolution layer, the output from the convolution layer contains channels with uniform values. When passed through a normalisation layer such as Batch Normalisation [47] or Instance Normalisation [48], the values in the channels become zero. Thus the semantic information is washed away when passed through a normalisation layer. To overcome this problem, a conditional normalisation method was introduced in [40], which is SPADE.

SPADE stands for Spatially Adaptive Normalization. In the proposed network, the Batch Normalisation Layers are replaced with SPADE Normalisation in all the layers of the generator, which is a ResNet[49]. The principle behind SPADE is that instead of learning the

affine parameters in the Batch norm, the semantic maps are used to compute the affine parameters. Figure 2.15 shows the affine parameter calculation in SPADE.

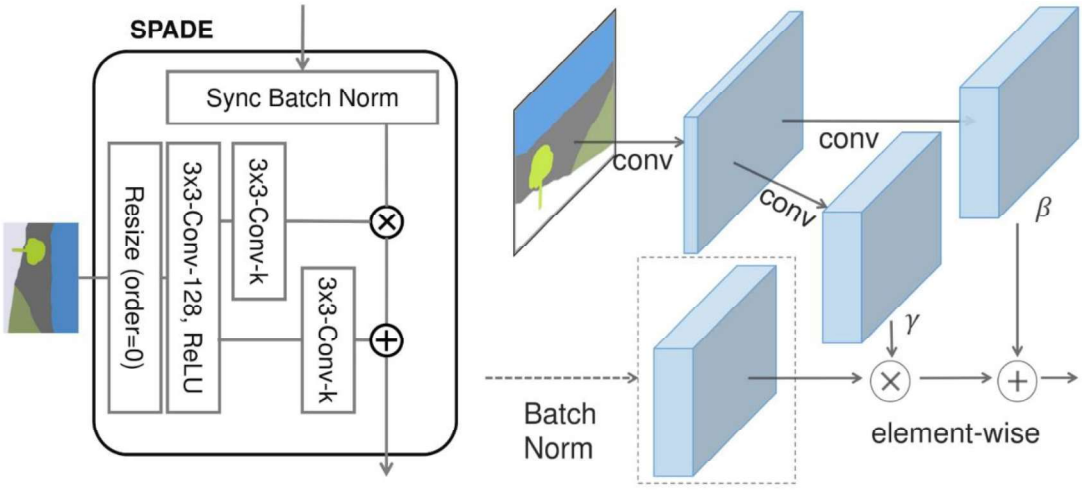


Figure 2.15: SPADE normalization. Source: [40]

Since the affine parameters are learned based on the semantic maps, the SPADE is a conditional norm. Hence, using SPADE, the semantic information is propagated effectively throughout the network which helps generate meaningful images even when the input semantic map consists of only a single label class.

Typically, the generator network in conditional GAN is an encoder-decoder where the input image is provided to the encoder, and the decoder generates an image, based on the encoded input image. Whereas, in SPADE, the generator network consists of only a decoder since the semantic map is not explicitly provided as input. Instead the semantic map is given

to each layer in the generator network. Since the encoder is eliminated, the model is more compact with less parameters. A Gaussian noise image is given as input to the decoder network. The network architecture of SPADE is shown in Figure 2.16.

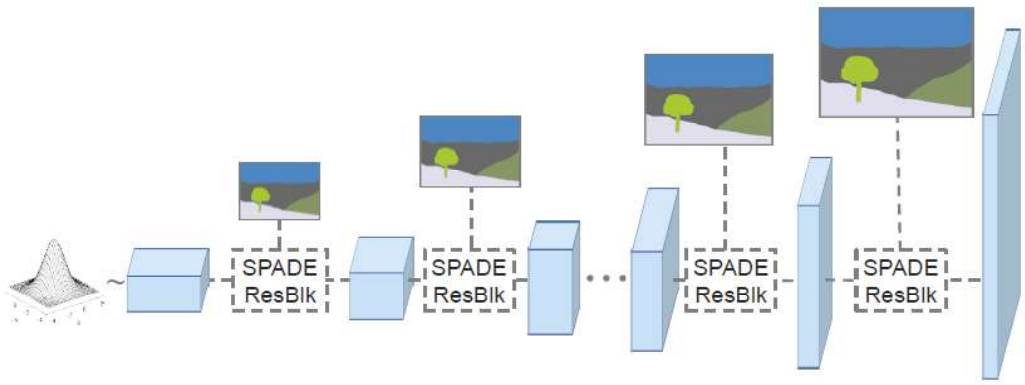


Figure 2.16: Network architecture used in [40]

We evaluated the bark color generation using both Pix2pixHD and SPADE, and found that SPADE generated better quality results compared to Pix2pixHD. Similar to pix2pix, we use images of size 256×256 for both pix2pixHD and SPADE. Our dataset consists of plenty of label maps with a single class, and SPADE generated meaningful images for those labels. An example is given in Figure 2.17. When a semantic map having a single class is provided as input to the networks, pix2pixHD produces a blank image since the semantic information is lost after the normalization layers. Whereas, SPADE generates an image bark.

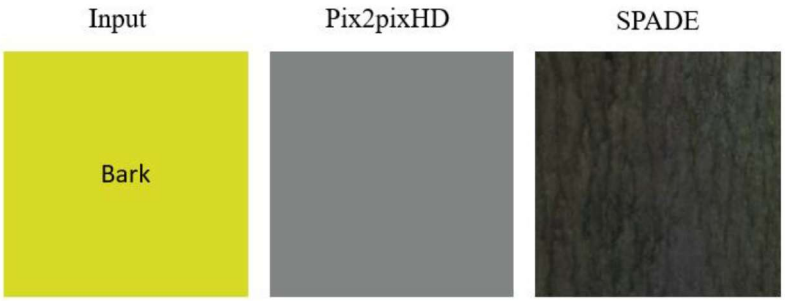


Figure 2.17: When the input semantic consists of only a single class, pix2pixHD does not produce any meaningful result due to the loss of semantic information after the normalization, while SPADE generates the bark.

While generating the bark colors using SPADE, we observed that the generated images were deterministic. In other words, the network generated a particular image for a given label. Hence, every time an input label with a single class was provided, the network generated the same image with no variations. Providing the generated radius map along with the semantic map to the generator network solved this problem. Even though the semantic maps are identical, the radius maps are unique. Thus, the network also generated unique color barks. The other method was to add noise to the input semantic maps. Adding noise to the semantic maps yielded only minimal variations in the generated images. Hence, we used the radius maps to improve the multimodality of the network. Some of the oak barks generated after achieving the multimodality is shown in Figure. 2.18. Even though the input semantic map has only a single class “bark”, the network generates different patterns of the oak bark in the color images, due to different radius maps provided as input. Whereas, this was not the case when only the semantic map was provided as input, which generated only a single deterministic color bark for the same semantic map.

Figure 2.18: Improved multimodality by using radius maps to generate different oak barks for the same input semantic

2.6.4 Generating the maps concurrently

One drawback of using SPADE for our objective is that the SPADE network can only be used when the input semantic maps are provided. While semantic maps are useful to generate the bark colors, they are not required for the radius generation. This means, the radius must be generated by a different network (pix2pix), and the bark color by SPADE. Instead of having two separate networks, we wanted a simpler pipeline by generating both the radius and the bark color using a single network. Also, the bark color generation depends on the radius map to generate multi-modal outputs. Hence, we wanted to develop a faster and a more compact network that generates both the radius and the color images simultaneously. Thus, we developed an architecture based on SPADE called the Multimodal SPADE, that uses the input smooth approximate radius map to generate both the radius and the bark color. Our architecture is made up of a generator network and two discriminator networks. The generator consists of an encoder and two decoders: radius generator and color generator. One of the discriminator networks is used to discriminate the radius images, while the other is used for the color images.

The radius generator generates the radius maps, while the color generator generates the bark color. The encoder and decoder networks are made of ResNet blocks. The network architecture of Multimodal SPADE is shown in Figure 2.19 and Figure 2.20. The input to the encoder network is the smooth radius image. The encoder network encodes the input into a

latent space. The decoder then uses the latent space obtained from the encoder to generate the radius maps and the color.

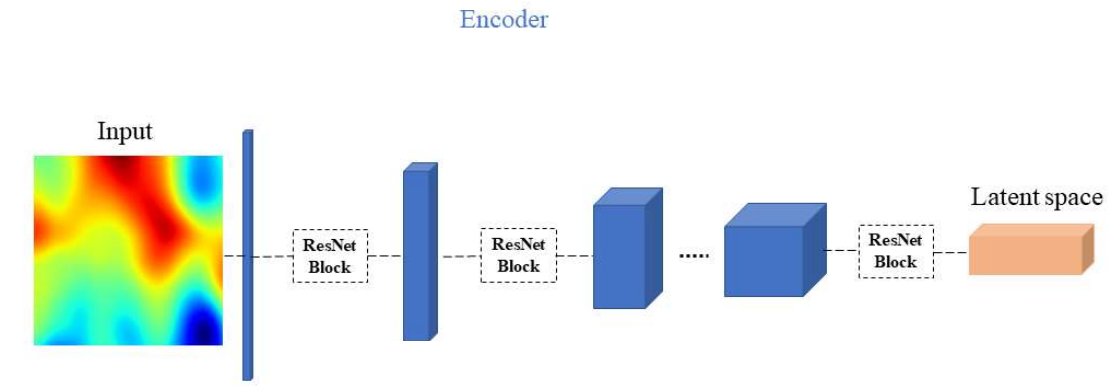


Figure 2.19: The encoder network of Multimodal SPADE to convert input smoothened radius to a latent space

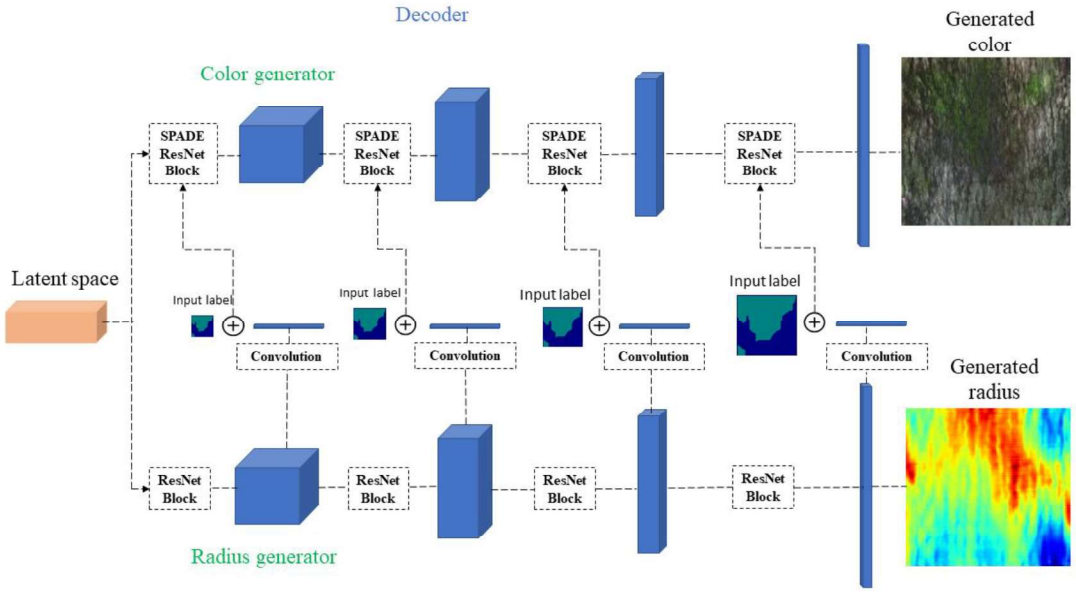


Figure 2.20: The decoder network of Multimodal SPADE which generates both the radius and color

The radius generator is made of ResNet blocks. We don't use semantic maps in the radius generator, so each ResNet block consists of the standard Batch Normalization layers.

The network of the color generator is similar to the one used by SPADE architecture. The semantic maps are provided to the generator through the SPADE normalisation. In order to ensure multi-modality of the color generator, the output channels from each of the radius generator's ResNet blocks are passed to the color generator. Initially, all the channels from the radius generator were passed to the color generator. But our experiment showed that doing so caused the semantic maps to be overpowered by the radius channels. As a result, the color generator started generating moss at arbitrary places, similar to the color images generated by pix2pix. To maintain a balance between the radius input and the semantic input, we had to limit the number of channels passed to the color generator from the radius generator. To limit the number of channels, we pass the radius generator's channels through a convolution layer and appended the channels to the input semantic map. This is given as input to the corresponding color generator's SPADE ResNet blocks. We found that this yielded better results, where we could control the location of moss and defects and at the same time achieve multi-modality.

We use hinge loss, perceptual loss[50] and feature matching loss for both the radius and color generators.

The **hinge loss** is given by:

$$L_D = E_{(x,y)\sim p_{data}}[max(0, 1-D(x,y))]+E_{z\sim p_z, y\sim p_{data}}[max(0, 1+D(G(z,y),y))]$$

$$L_G = -E_{z\sim p_z, y\sim p_{data}}[D(G(z,y),y)]$$

where, L_D is the discriminator loss and L_G is the generator loss, x is the training data, y is the generated image, z is the input latent vector. The functions G and D represent the output from the generator and discriminator respectively.

The **perceptual loss** was developed in place of per-pixel loss. Perceptual loss functions are used when comparing two different images that look similar, like the same photo but shifted

by one pixel. The function is used to compare high level differences, like content and style discrepancies, between images.

Feature matching loss causes training to be more stable and helps converge faster. Feature matching changes the cost function for the generator to minimizing the statistical difference between the features of the real images and the generated images. Therefore, feature matching expands the goal of the generator from beating the opponent to matching features in real images.

We used two multi-scale discriminators in our network: one to discriminate the radius images and the other for the color images. The architecture of the discriminator is same as the one used in SPADE.

2.7 Tiling the images

The images obtained from Multimodal SPADE are of the size 256x256. To build the full structure of a tree trunk, the images must be combined together to form a full map. When the image tiles are combined, there is bound to be a discontinuity at the places where the images are joined. These discontinuities must be removed to obtain smoothly varying maps. An existing GAN architecture for tiling images is TileGAN[51]. However, it requires that the maps be generated by another GAN network ProgressiveGAN[32], which does not depend on the input image and instead generates image tiles on its own. Moreover, the TileGAN network modifies many parts of the original image for tiling, which is not desirable for our application.

Instead, we used an inpainting method, where the discontinuities in the tiled image are masked and a conditional GAN network was used to generate a continuous image by filling only the masked regions. Edge-Connect[42] is an inpainting network, that has two generator networks, one to connect the missing edges and the other for filling colors. However, it uses Pix2pixHD for inpainting. Since Pix2pixHD network was found to produce color scale

variation between the input and the generated image for our dataset, which destroys the original radius values while generating the radius maps, we developed a variant of the pix2pix architecture to tile the images. The network architecture is shown in Figure 2.21.

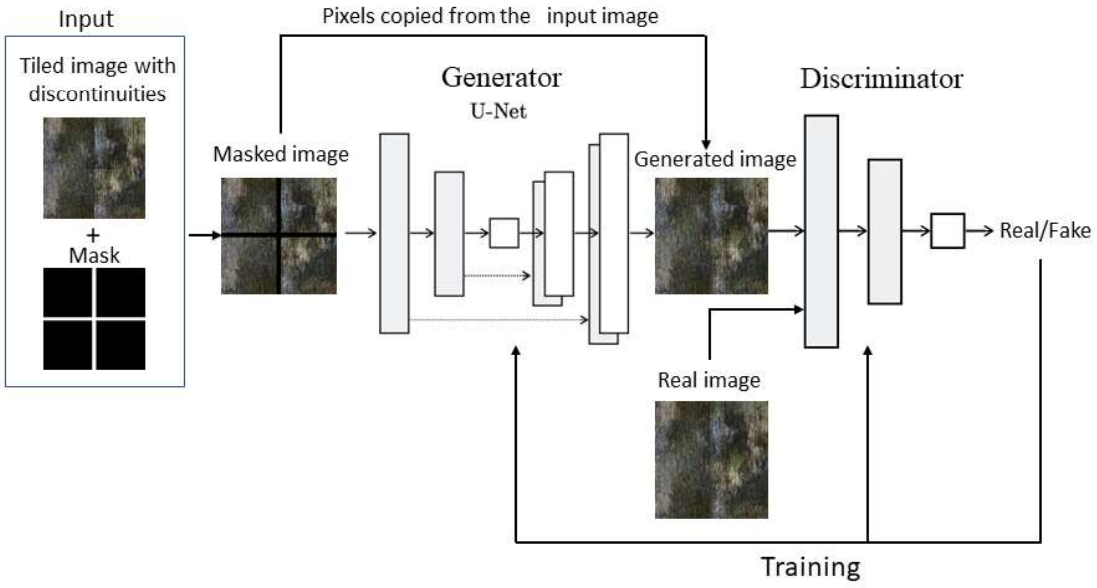


Figure 2.21: GAN architecture for tiling images

To tile the images, we combine four image tiles into one. Since our individual tiles are 256×256, the resolution of the image after combining the tiles is 512×512. The combined images will have a discontinuity at the places where the images are joined. The discontinuities are masked. The masked areas were filled with the mean values of the image. Doing this gave us better results and helped achieve convergence faster. The GAN tends to modify the parts of the images other than the places that needs to be filled. This is especially detrimental while combing the radius tiles since the original values would be replaced. To avoid this, we propose an architecture, based on pix2pix.

In the modified pix2pix architecture, the generated image is copied with the unmasked pixels from the input image, and this image is given to the discriminator. This way, the generated images were of the same quality as the original image. This ensured that even though the generator generates higher resolution images, the quality of the generated images is not affected. Without copying the pixels from the original image, the generated images were slightly different from the input image and had some artifacts. Tiling was done for both the radius and color images to obtain the full maps.

While tiling the color images obtained from pix2pix, we found that most of the images had moss spread uncontrolled. Because of this, the image doesn't look like a realistic bark. There were sharp variations in bark colors all over the images generated by pix2pix. The network had learnt to generate them due to the illumination variations caused by shadows and sunlight in our training dataset. To homogenize the colors, we used GP-GAN[17] network. The GP-GAN was originally designed to blend two images. The GP-GAN network homogenises the color variations in the image, while maintaining the high frequency details in the image. This makes sure that the gradients are retained while only the color is modified. The result after applying GP-GAN is shown in the Figure 2.22. Although the colors are homogenized, the image looks blurry. Also, using GP-GAN does not help us place the moss and defects at our desired locations. This was also one of our motivations to use SPADE to generate the color images.

Figure 2.22: Tiled image with homogenized color

2.8 Creating 3D meshes

The radius and color maps obtained were used to create 3D models of the trees. The surface geometry of the tree bark was obtained from the radius map. The radius maps contain the radius information as a function of the height of the tree and angle i.e it contains the information of the 3D point cloud of the tree bark in polar coordinates. However, it is much easier to work with Cartesian co-ordinate. Thus, the polar co-ordinate points were converted to Cartesian coordinates. The resulting point cloud was converted to a solid mesh using triangle strips in VTK[52]. The color information was also added to obtain the final 3D model of the tree bark. A 3D model of an oak bark without and with color is shown in Figure 2.23.

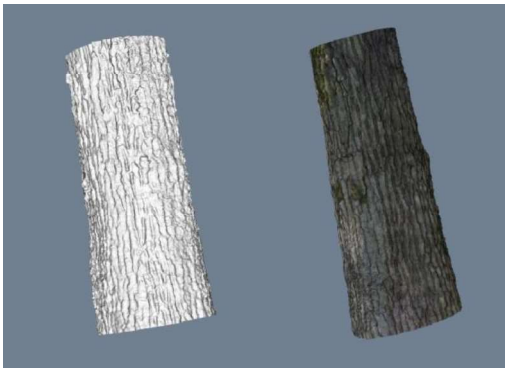


Figure 2.23: 3D meshes of an oak bark without and with color

CHAPTER 3

EXPERIMENTS

3.1 Introduction

During the study we conducted many experiments to determine the best approach to meet our objective. We also explored different architectures of GANs to generate the 3D models of fake trees. To overcome the shortcomings of the existing methods, we also proposed our own solutions, which were explained at relevant sections in Chapter 2. In this chapter, we provide in detail, the various approaches that we considered for our objective.

3.2 Dataset Acquisition for 3D Reconstruction

Initially, the videos of oak trees were acquired and the entire pipeline was tested for it. The videos were acquired on a sunny day with no additional lighting setup. Hence most of the oak trees in our dataset consists of regions with shadows and bright light mixed. Because of this, the color maps that were extracted from the reconstructed point cloud consisted of regions with varying color intensities. This in turn, affected the quality of color maps generated by the GANs. The GANs, generated bark color images with varying intensity at random places, which created a unrealistic effect.

Considering this, we acquired the videos of the beech trees on an overcast day. The bark colors were free from drastic changes due to shadows. This resulted in more realistic generation of bark images from the GANs. Hence, while testing the proposed pipeline, it is desirable to acquire the datasets under uniform lighting conditions to obtain good results. After acquiring the images of the trees, the background parts of the images were suppressed.

3.3 3D Reconstruction of real trees

Once the background-suppressed images of the trees were obtained, they could be used to perform multi-view 3D reconstruction to obtain the 3D point clouds. To generate the 3D point clouds we considered Pix4DMapper and Colmap. Pix4DMapper could generate high quality point clouds in a much shorter time compared to Colmap. For each tree, Colmap took around 20 hrs to generate the dense point cloud, whereas it was around 45 minutes in the case of Pix4DMapper. Pix4DMapper could also handle more input images compared to Colmap. To test the quality of 3D reconstruction, we performed a quantitative comparison between the point clouds obtained from the photogrammetry softwares with the ground truth, as explained in the next section.

3.4 Numerical Comparison of point clouds

The quality of the point clouds obtained from Colmap and Pix4DMapper were assessed by comparing the point clouds with the ground truth. The ground truth was obtained using a laser scanner. For comparison, the mid-section of the trunk was used.

Comparison of the point cloud obtained from Colmap and Pix4DMapper with the point cloud from laser scanner was done using CloudCompare[53]. To compare two point clouds, the following steps must be performed:

- Scaling the point cloud
- Registering the point clouds
- Measurement of distance between the point clouds

3.4.1 Scaling the point cloud

Initially, the point cloud obtained from the 3D reconstruction was scaled to match the dimensions of the ground truth point cloud. To perform the scaling, two specific points that are

clearly visible in both the point clouds are selected and the distance between the points are measured. This way, the exact dimension can be obtained from the ground truth and the scaling factor for scaling the reconstructed image can be calculated.

Images acquired using monocular cameras cannot capture the global scale information. Also, we do not have the ground truth point cloud for each of the 3D reconstructed tree. To get a reliable source of scale information of the 3D reconstructed trees, we attached markers of known dimension onto the lower parts of the tree bark. Thus after 3D reconstruction is performed, we scale the reconstructed trees to match the known dimension of the marker.

3.4.2 Registering the point clouds

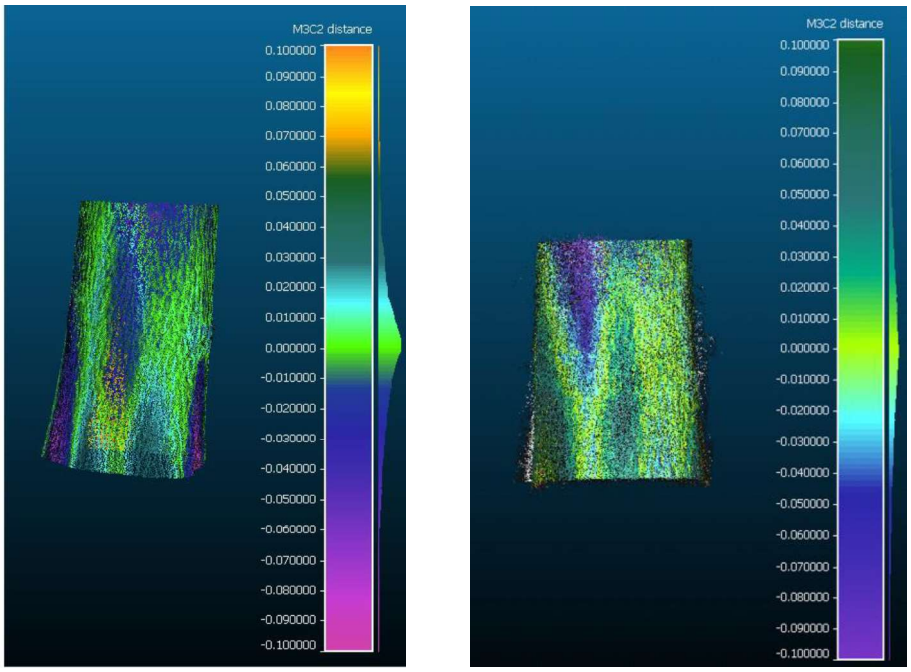
After scaling, both the point clouds for comparison will be of the same dimension. After this, both the point clouds are aligned together. This is done by Iterative Closest Point (ICP), where one of the point cloud is rotated and translated relative to the other point cloud so that both the point clouds are aligned.

3.4.3 Measuring the distance between the point clouds

After aligning the point clouds together, the distance between both the point clouds are calculated for each individual points. The metrics for comparison could be L1 norm, L2 norm, Hausdroff, etc.

The comparison showed that the majority of points have an error of zero mean and a standard deviation of 1cm. The comparison for the point clouds obtained from Colmap and Pix4dMapper with the point cloud obtained from laser scanner is shown in Figure 3.1. The histogram from the images show that significantly more points are concentrated at zero mean when the reconstruction is done using Pix4DMapper, than when using Colmap. From the results, it can be concluded that the quality of 3D reconstruction is much better in

Pix4DMapper. Considering the factors such as the quality and speed of reconstruction, and the ability to handle more images, we chose Pix4DMapper to perform our 3D reconstructions.



(a) (b)

Figure 3.1: Comparison of point cloud from laser scanner with point cloud from (a) Pix4dMapper (b) Colmap. From the histogram in (a) and (b), it can be seen that majority of points have a zero mean in the case of Pix4DMapper, showing that Pix4DMapper yields a better quality reconstruction.

3.5 Experiment on GANs

3.5.1 Dataset

Once the radius and color maps were extracted from real tree point clouds, they were used to train the GAN networks to generate fake samples of the maps. Since GANs could only handle small images, the original radius and color maps were split into smaller tiles of resolution 256×256. The entire set of radius images were normalized to the range 0 to 65535 for training. We observed that without normalizing the images, the network could not learn the mapping between the input and the output properly and generated poor quality images. We

were also limited by the number of radius and color maps obtained from the original trees. To train a GAN, it is important to provide it with lot of training examples to achieve a wide range of variability in the generated images. To augment our training dataset, we stride over the original maps with a stride rate of 64 to create the smaller tiles. The training set for the oak tree consists of 2088 images and the test set consists of 344 images. The training set for the beech tree consists of 1192 images and the test set consists of 256 images.

3.5.2 Sematic maps for SPADE and Multimodal SPADE

Apart from the image radius and color image tiles, the SPADE and the Multimodal SPADE networks require input semantic maps for training. The semantic maps for the tree barks were obtained by hand labelling each of the bark images. An online labelling tool Labelbox [54] was used. Each of the bark color images were labelled into four categories, and are represented with the following colors in the semantic maps:

1. Bark - Yellow
2. Defect - Orange
3. Moss - Red
4. Lichens - Blue

Samples of semantic maps and the corresponding bark images from our dataset are shown in Figure. 3.2 and Figure 3.3.

Similar to the radius and color image maps, the semantic maps were also split into sizes of 256×256 and we used a stride rate of 64 to split the image tiles and augment our dataset.

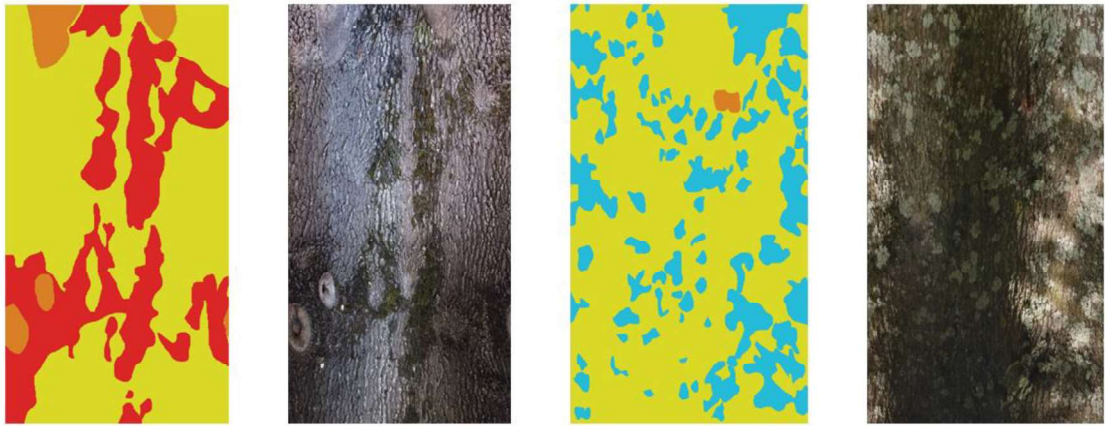


Figure 3.2: Oak bark with the corresponding semantic map

Figure 3.3: Beech bark with the corresponding semantic map

3.5.3 Implementation details of pix2pix

We used the original parameters proposed in [33] for our experiments. The learning rates of the generator and discriminator were 0.0002. We used the Adam solver with $\beta_1 = 0$ and $\beta_2 = 0.999$. The experiments were conducted on an 8 GB Nvidia GeForce GTX 1080. The training took about 5 hours, for a total of 200 epochs.

3.5.4 Implementation details of SPADE and Multimodal SPADE

We used the original parameters proposed in [40] for our experiments. The learning rates of the generator and discriminator were 0.0001 and 0.0004 respectively. We used the Adam solver with $\beta_1 = 0$ and $\beta_2 = 0.999$. The experiments were conducted on a 12 GB Nvidia GeForce GTX 1080. The training took almost two days for 200 epochs.

CHAPTER 4

RESULTS

4.1 Introduction

In this chapter, we provide the results of the experiments conducted as part of the pipeline. We present the results in two sections. In the first section, we show the results for the 3D reconstruction and surface geometry extraction. In the second section, we provide the results of GANs and finally, the 3D models of fake oak and beech tree barks obtained using our pipeline.

4.2 3D Reconstruction and Surface Geometry Extraction

After acquiring the images, the background portions were removed to improve the quality of the tree being reconstructed and to obtain a faster reconstruction. An example of a tree before and after background suppression is shown in Figure 4.1.

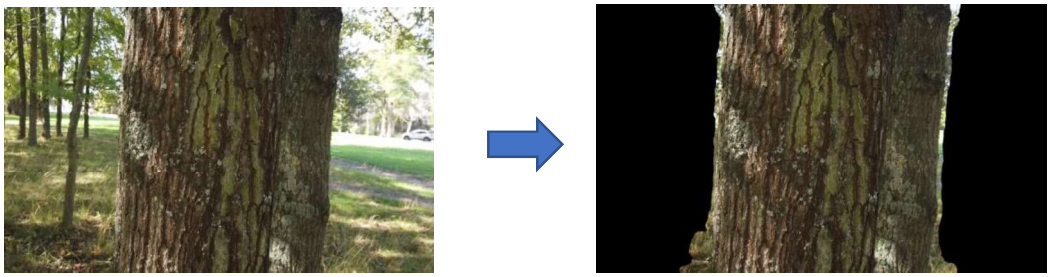


Figure 4.1: An oak tree before and after background suppression

After performing image suppression, the images were used by Pix4DMapper to perform 3D reconstruction of the trees. The result of 3D reconstruction is the 3D point cloud of the

trees. Images of some of the 3D point clouds of oak and beech tree are shown in Figure 4.2 and Figure 4.3.

Figure 4.2: 3D point clouds of oak trees

Figure 4.3: 3D point clouds of beech trees

From the 3D point cloud, solid 3D meshes can be created. Two of the commonly used methods for meshing are Poisson meshing and Delaunay meshing. A 3D mesh of an oak tree was constructed using Poisson meshing, shown in Figure 4.4. From the mesh, one could visualize the ridges of an oak tree. This shows that the 3D reconstruction was able to capture the nuances on the bark structure. The Poisson meshing feature is in-built in both Colmap and Pix4dMapper.

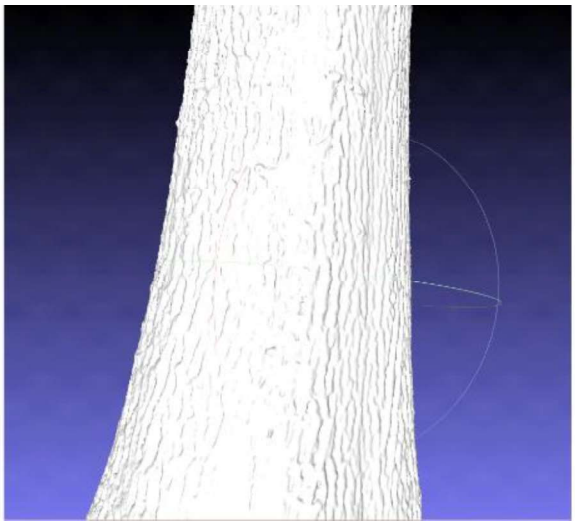


Figure 4.4: 3D Mesh of an oak tree

Once the 3D point clouds of the trees were acquired, the radius and color information were extracted from them using the method proposed in Section 2.5. To reiterate, the radius maps consists of the geometry of the tree bark in polar co-ordinates, where, the map contains the radius values from the centre points as a function of the height of the tree and the angle. This is similar to cutting open a 3D cylinder and spreading it into a rectangle in 2D. The radius maps of oak trees are shown in Figure 4.5 and the corresponding bark colors in Figure 4.6. The radius maps of beech trees and the corresponding bark colors are shown in Figure 4.7 and Figure 4.8. The radius maps obtained are shown in jet colormap for visualisation.

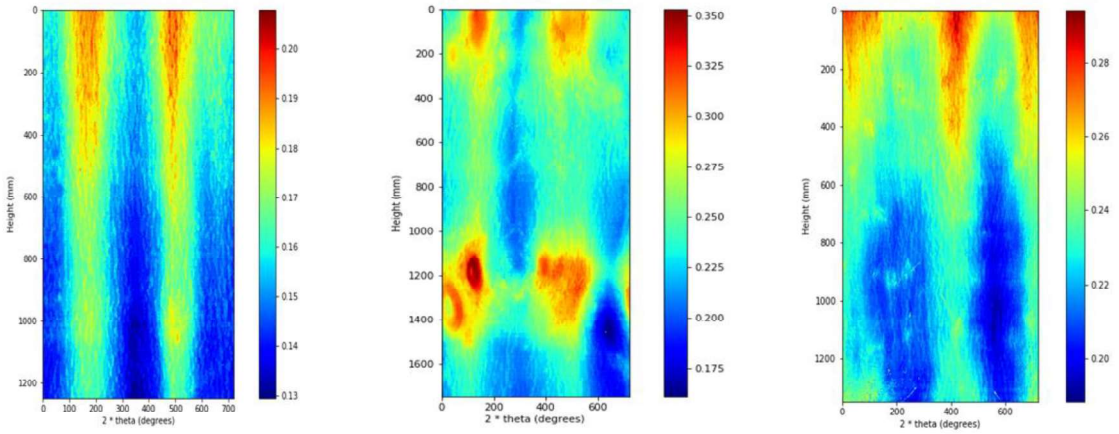


Figure 4.5: Radius maps of oak trees

Figure 4.6: Oak Barks

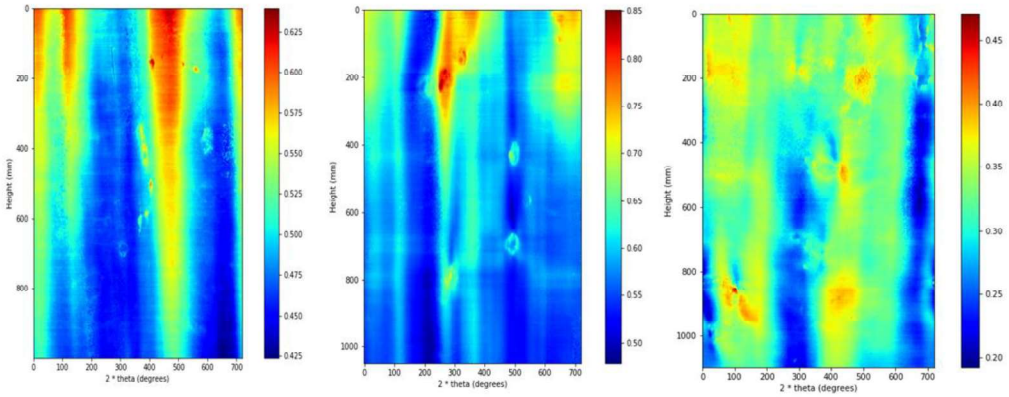


Figure 4.7: Radius maps of beech trees

Figure 4.8: Beech barks

4.3 Generative Adversarial Networks

4.3.1 Architecture

In this section, we compare the results obtained from pix2pix, SPADE and Multimodal SPADE for the generated radius and color maps. To evaluate all the architectures, we trained all the networks using the same dataset for both oak and beech trees and for the same number of epochs. The comparison results for oak and beech trees are shown in Figure 4.9 – Figure 4.12.

The results show that majority of the radius maps generated by Multimodal SPADE are structurally more defined, when compared to those generated by pix2pix. While the pix2pix generated radius maps are more blurry, the Multimodal SPADE generated maps have clearly defined features, which is evident, especially for oak barks. The results for bark color show that pix2pix generated images have moss, lichens and barks mixed up randomly. The bark images generated by SPADE and Multimodal SPADE are more constrained, where the moss, defects, bark and lichens are present only at places defined in the input semantic maps. While SPADE generates the identical bark patterns when the input semantic map has a single class, Multimodal SPADE can generate unique bark images even when the input semantic has only a single class. One other distinction that could be observed between the images generated by SPADE and Multimodal SPADE is that the bark patterns are clearly visible in Multimodal SPADE. This is because, for each layer in the network, the radius maps are also passed to the color generator. Thus the network learns to include the bark patterns in the color images. Whereas, the bark patterns are not well-defined for SPADE generated bark images. These results show the advantage of Multimodal SPADE over the other methods for generating the radius and color maps.

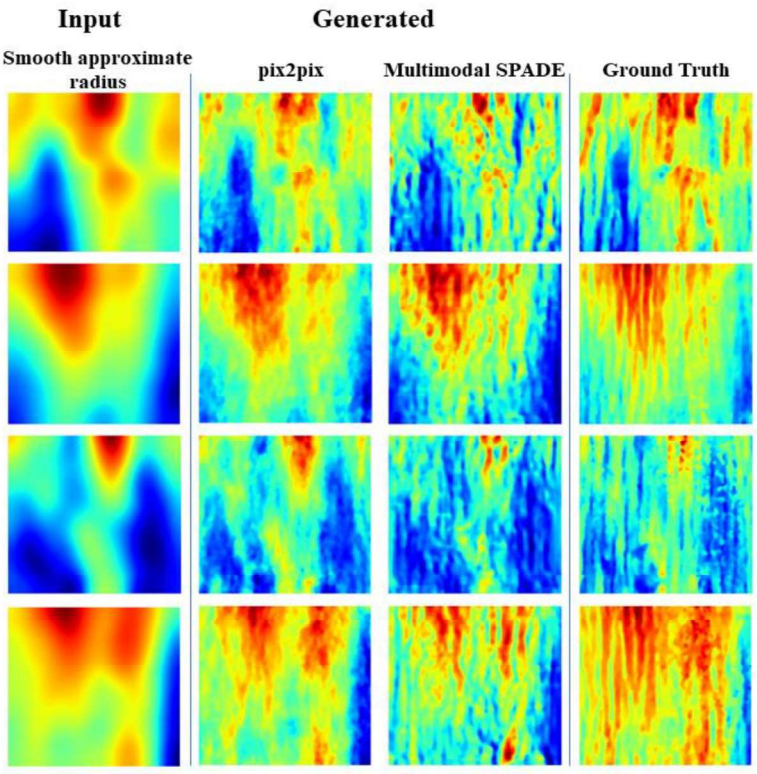


Figure 4.9: Different methods to generate radius maps for oak trees

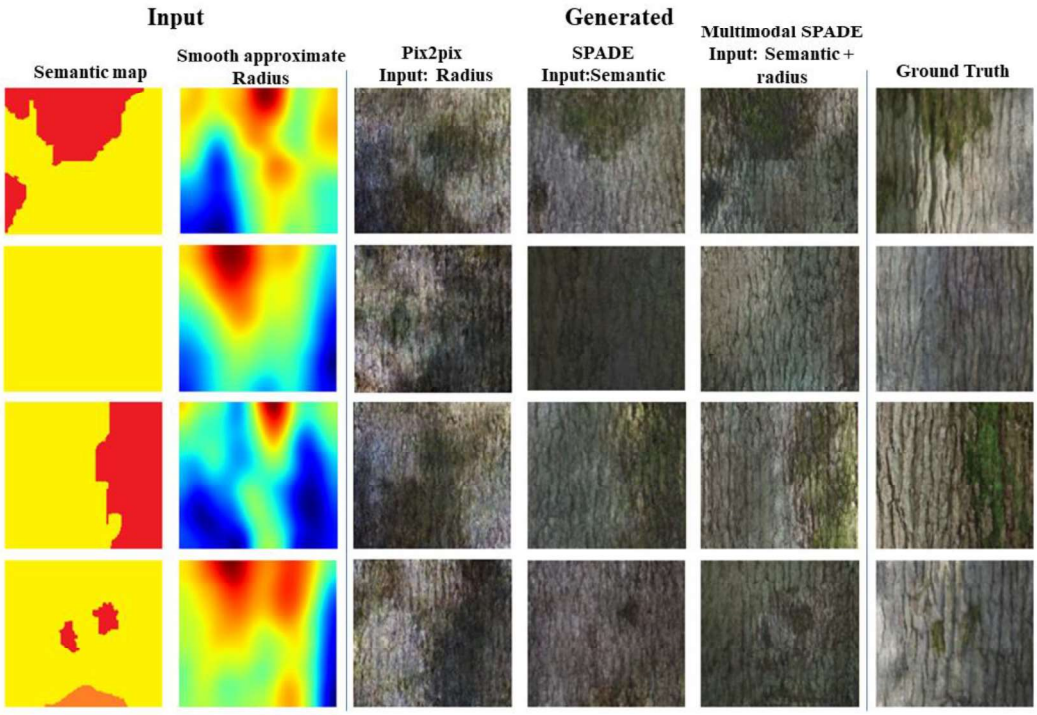


Figure 4.10: Different methods to generate oak bark colors

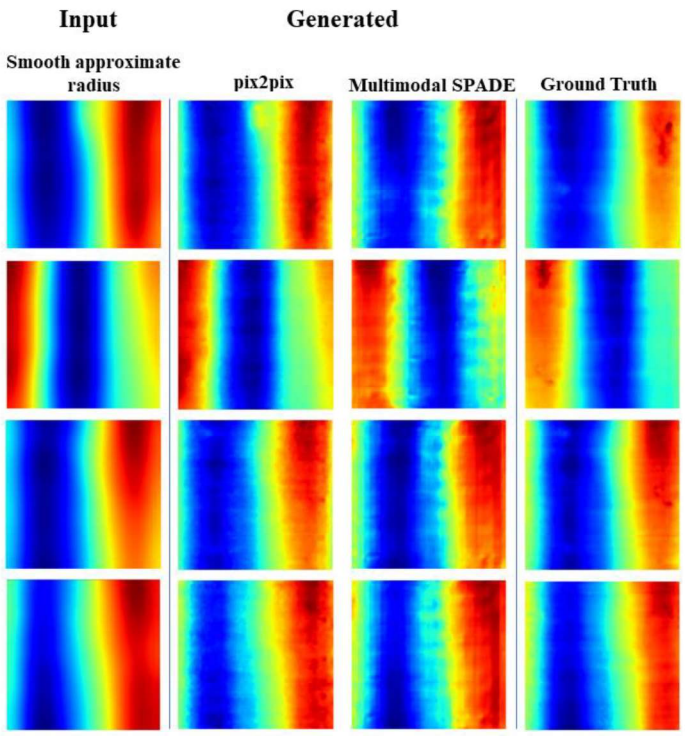


Figure 4.11: Different methods to generate radius maps for beech trees

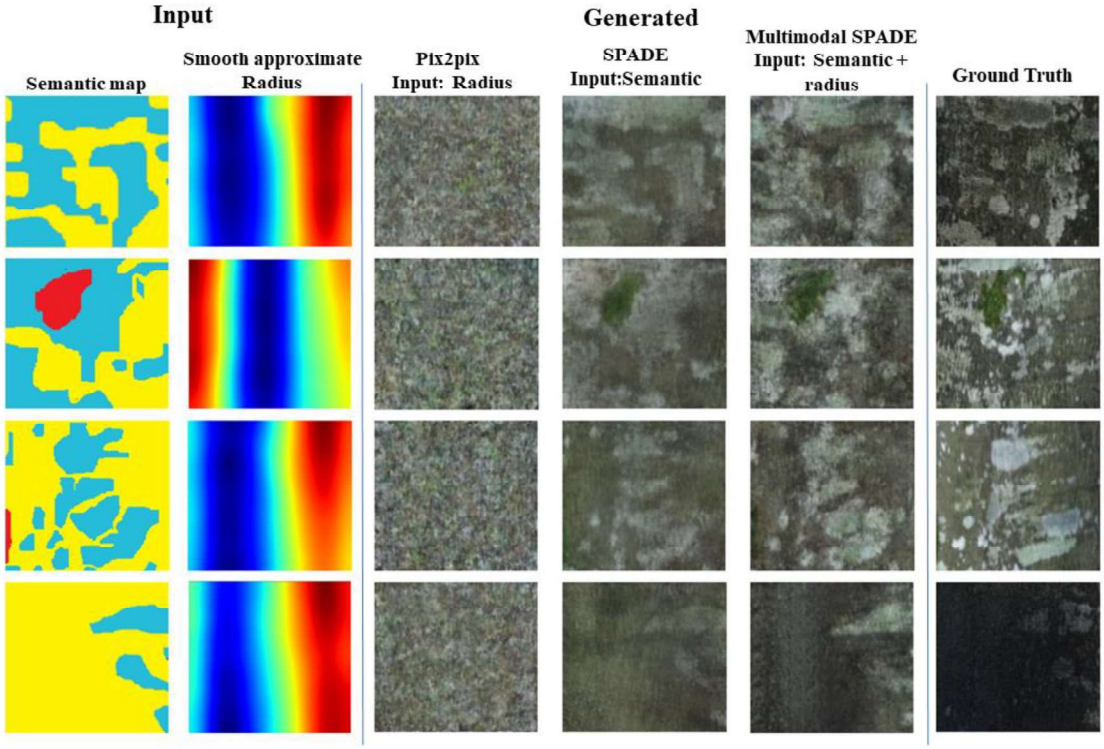


Figure 4.12: Different methods to generate beech bark colors

4.3.2 Tiling the images

The generated images from the GAN networks were tiled using our modified pix2pix architecture. An example continuous image obtained by combining four smaller color images of an oak bark is shown in Figure 4.13. The discontinuities at the point of intersection of the images, which can be observed in Figure 4.13(a) was masked, as shown in Figure 4.13(b). The modified pix2pix network then filled the regions to obtain a continuous image as shown in Figure 4.13(c). A full continuous bark image of oak and beech obtained after tiling is shown in Figure 4.14.

(a) (b) (c)

Figure 4.13: (a) The tiled image with discontinuities (b) Image after discontinuities masked (c) Continuous image obtained from the tiling network

(a) (b)

Figure 4.14: Full continuous maps of oak and beech tree barks obtained after tiling

4.3.3 Construction of the 3D meshes

After the tiled radius and the corresponding color maps were obtained, they were used to construct 3D meshes using the procedure explained in Section 2.7. The 3D models of oak and beech trees without and with the color are shown in Figure 4.15 - Figure 4.17. From the 3D mesh without color, the finer structural details of the tree barks can be observed. The GANs have captured the nuances on the oak and beech barks, and the fake 3D models look as realistic as the original 3D model. For the 3D mesh along with the bark color, the pix2pix generated model for the oak bark looks homogenous, since GP-GAN was used to homogenize the colors. However, using GP-GAN, one cannot retain the moss and defects on the bark surface, hence, not much variability in the bark color can be obtained while using pix2pix. The beech bark color generated by pix2pix doesn't look realistic since it has moss and lichens randomly mixed together. In comparison, the oak and beech barks obtained using Multimodal SPADE looks

very realistic. Since there is more control over the bark color generated, it results in realistic looking 3D models.

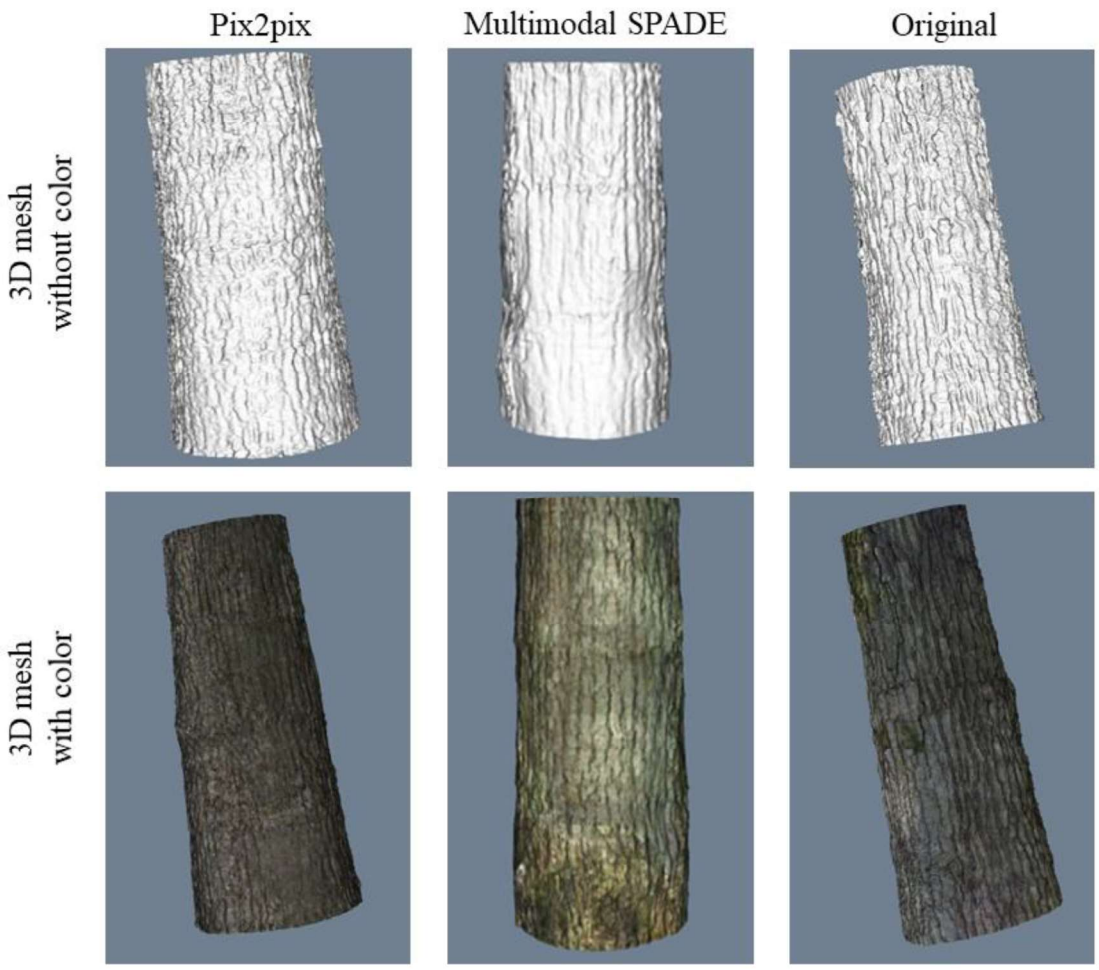


Figure 4.15: 3D models of oak trees without and with color

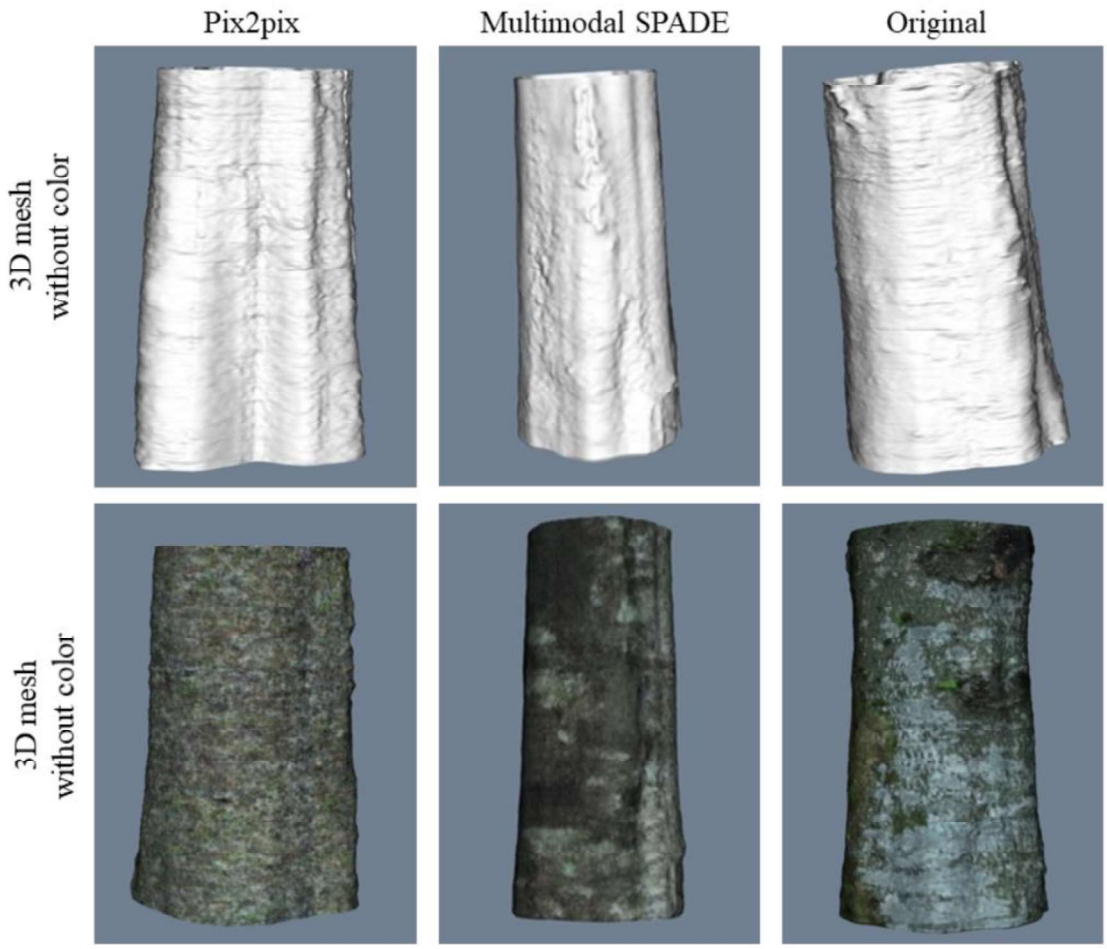


Figure 4.16: 3D models of beech trees without and with color

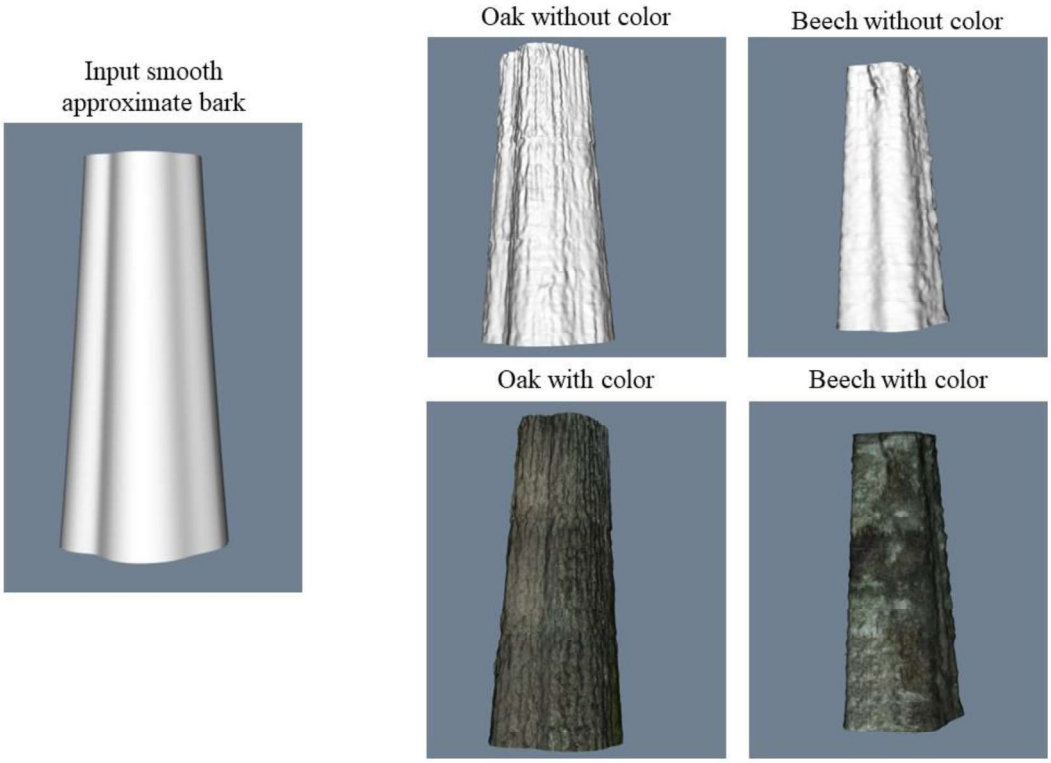


Figure 4.17: 3D models of fake oak and beech bark generated from a smooth approximate radius map

CHAPTER 5

CONCLUSION

In this work, we propose a novel pipeline to generate realistic tree barks from a set of images. Deviating from the conventional methods, we use deep learning to generate the tree barks. Our pipeline was initially developed for oak trees, and later extended for beech trees. We illustrated an efficient method to suppress the background portions of the image sequences to yield a better quality tree model and at the same time, reconstruct the trees faster. We also presented a method to tile smaller images obtained from GAN to produce a continuous map of the surface and color of the tree barks.

We also proposed a new GAN architecture called Multimodal SPADE to constrain the location of bark defects and moss. Instead of using two separate GAN networks to generate the radius and color, Multimodal SPADE can generate both simultaneously, thus saving time and computational resources. Our network uses the radius to generate unique bark patterns in the generated color, thus achieving multimodality.

Our method was successfully tested on trees with smooth and ridged barks, which yielded high-quality tree barks. Our method could be easily extended to other tree bark types to obtain genuine looking barks.

In the future, one direction of research could be to constrain the GAN further to have more control over the generated bark. Currently, the network generates defects at user defined places, however there is no control over the type of defect. This could be done for both the

radius and color maps. Similarly, the oak barks in our dataset consists of bark colors of various shades, ranging from off-white to dark brown. At present, we do not have any control over the color shade of the bark. Thus there is a possibility for the GAN to generate barks of two different shades for the same tree. This would lead to unrealistic effect while tiling the images. The GAN network could be constrained to generate bark colors of a particular shade chosen by a user. This way the user could have more control over the type of bark generated. However, the above methods would require a large amount of dataset for training the GANs.

REFERENCES

1. Ke Xie, Feilong Yan, Andrei Sharf, Oliver Deussen, Baoquan Chen, and Hui Huang, "Tree modeling with real tree-parts examples," IEEE Transactions on Visualization and Computer Graphics, vol. 22, pp. 2608–2618, December 2016.

2. X Chen, Y Neubert, Q Xu, O Deussen, and S.B Kang, "Sketch-based tree modeling using markov random field," ACM Transactions on Graphics, pp. 109:1–109:9, 2008.

3. M Okabe, S Owada, and T Igarashi, "Interactive design of botanical trees using freehand sketches and example-based editing," Computer Graphics Forum, pp. 487–496, 2005.

4. J Bloomenthal, "Modeling the mighty maple," in ACM SIGGRAPH, 1985.

5. Xi Wang, Lifeng Wang, Ligang Liu, Shimin Hu, and Baining Guo, "Interactive modeling of tree bark," in IEEE Pacific Conference on Computer Graphics and Applications, 2003.

6. Petr Laitoch, "Procedural modeling of tree bark," Bachelor thesis, Charles University, Prague, 2018.

7. Davison AJ, Reid ID, Molton ND, Stasse O, "MonoSLAM: Real-Time Single Camera SLAM. In IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 29, No. 6, pp. 1052–1067

8. Klein G, Murray DW, "Parallel tracking and mapping for small AR workspaces". In Proceedngs of International Symposium on Mixed and Augmented Reality. pp 225–234, 2007.

9. Mur-Artal R, Montiel JMM, Tardós JD "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In IEEE Transactions on Robotics 31(5):1147–1163. doi:10.1109/TRO.2015.2463671, 2015.

10. Mur-Artal R, Tardós JD, "ORB-SLAM2: An Open-source SLAM System for Monocular, Stereo and RGB-D Cameras". In CoRR. abs/1610.06475, 2016.

11. Newcombe RA, Lovegrove SJ, Davison AJ, "DTAM: Dense Tracking and Mapping in Real-Time". In Proceedings of International Conference on Computer Vision(ICCV), 2011, pp 2320–2327.

12. Engel J, Schöps T, Cremers D, "LSD-SLAM: Large-scale Direct Monocular SLAM". In Proceedings of European Conference on Computer Vision(ECCV), 2014, pp 834–849.

13. Engel J, Koltun V, Cremers D, "Direct Sparse Odometry". In CoRR. abs/1607.02565, 2016.

14. Ummenhofer, B., Zhou, H., Uhrig, J., Mayer, N., Ilg, E., Dosovitskiy, A., Brox, T. “DeMoN: Depth and Motion Network for Learning Monocular Stereo”. In IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2017.

15. Huizhong Zhou, Benjamin Ummenhofer, Thomas Brox. “DeepTAM: Deep Tracking and Mapping”. In European Conference On Computer Vision(ECCV), 2018.

16. Felix Endres, Jurgen Hess, Nikolas Engelhard, Jurgen Sturm, Daniel Cemers, Wolfram Burgard. An Evaluation of the RGB-D SLAM System. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2012.

17. Schonberger, Johannes Lutz and Frahm, Jan-Michael. Structure-from-Motion Revisited. In IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2016

18. Schonberger, Johannes Lutz and Zheng, Enliang and Pollefeys, Marc and Frahm, Jan-Michael. Pixelwise View Selection for Unstructured Multi-View Stereo. In European Conference on Computer Vision (ECCV), 2016

19. https://www.pix4d.com

20. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. In Advances in Neural Information Processing Systems (NIPS), 2014.

21. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems (NIPS), 2012.

22. Mehdi Mirza and Simon Osindero, “Conditional generative adversarial nets,” arXiv:1411.1784, 2014.

23. E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In Advances in Neural Information Processing Systems(NIPS), 2015.

24. S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In ICML, 2016.

25. X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. In ECCV, 2016.

26. M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. ICLR, 2016.

27. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In ICLR, 2016.

28. Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley. Least Squares Generative Adversarial Networks. arXiv preprint arXiv:1611.04076, 2016.

29. Martin Arjovsky, Soumith Chintala, Leon Bottou. Wasserstein generative adversarial networks. In International Conference on Machine Learning (ICML), 2017.

30. Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing Of GANs For Improved Quality, Stability, And Variation. In International Conference on Learning Representation (ICLR), 2018.

31. Tero Karras, Samuli Laine, Timo Aila. A Style-Based Architecture for Generative Adversarial Networks. In IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2019

32. Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. CoRR abs/1912.04958, 2019

33. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

34. Jun-Yan Zhu*, Taesung Park*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in IEEE International Conference on Computer Vision (ICCV), 2017.

35. Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim and Jaegul Choo. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2018.

36. Andrew Brock, Jeff Donahue and Karen Simonyan. Large Scale {GAN} Training for High Fidelity Natural Image Synthesis. In International Conference on Learning Representations (ICLR), 2019.

37. Brian Teixeira, Vivek Singh, Terrence Chen, Kai Ma, Birgi Tamersoy, Yifan Wu, Elena Balashova, and Dorin Comaniciu. Generating Synthetic X-Ray of a Person from the Surface Geometry. In IEEE Conference on Computer Vision and Pattern Recognition, 2018.

38. Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. In IEEE International Conference on Computer Vision (ICCV), 2017.

39. Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In IEEE Conference on Computer Vision and Pattern Recogonition (CVPR), 2018.

40. Taesung Park, Ming-Yu Liu, Ting-Chun Wang, Jun-Yan Zhu. Semantic Image Synthesis with Spatially-Adaptive Normalization. In IEEE Conference on Computer Vision and Pattern Recogonition(CVPR), 2019

41. Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, Bryan Catanzaro. Image Inpainting for Irregular Holes Using Partial Convolutions. In European Conference on Computer Vision (ECCV), 2018

42. Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z. Qureshi, Mehran Ebrahimi. EdgeConnect: Generative Image Inpainting with Adversarial Edge Learning. arXiv preprint, 2019

43. Huikai Wu, Shuai Zheng, Junge Zhang, Kaiqi Huang. GP-GAN: Towards Realistic High-Resolution Image Blending. In ACMMM, 2019.

44. Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia, "Pyramid scene parsing network," in Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 2017, pp. 2881–2890.

45. Mans Larsson, Erik Stenborg, Lars Hammarstrand, Torsten Sattler, Marc Pollefeys, and Fredrik Kahl, "A cross-season correspondence dataset for robust semantic segmentation," in CVPR, 2019

46. Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele, "The cityscapes dataset for semantic urban scene understanding," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3213–3223.

47. Sergey Ioffe, Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (ICML), 2015.

48. Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. arXiv:1607.08022, 2016.

49. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2015.

50. Justin Johnson, Alexandre Alahi, Fei-Fei Li. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In European Conference on Computer Vision (ECCV), 2016.

51. Anna Frühstück, Ibraheem Alhashim and Peter Wonka. TileGAN: Synthesis of Large-Scale Non-Homogeneous Textures. In ACM Transactions on Graphics (Proc. SIGGRAPH), 2019.

52. Will J Schroeder, Bill Lorensen, and Ken Martin, The visualization toolkit: an object-oriented approach to 3D graphics, Kitware, 2004.

53. CloudCompare (version 2.10) [GPL software]. (2019). Retrieved from http://www.cloudcompare.org/

54. Labelbox, "Labelbox," Online, 2019. [Online]. Available: https://labelbox.com

55. Herbert Bay, Tinne Tuytelaars, Luc Van Gool, “SURF: Speeded Up Robust Features”, In European Conference on Computer Vision (ECCV), 2006.

56. David G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoint”, In International Journal of Computer Vision, 2004.

57. E. Rosten and T. Drummond, “Machine learning for highspeed corner detection”, In European Conference on Computer Vision (ECCV), 2006.

58. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski, “ORB: an efficient alternative to SIFT or SURF”, In International Conference on Computer Vision(ICCV), 2011.